# Universal Hash Families 

Emin Karayel

March 12, 2024


#### Abstract

A $k$-universal hash family is a probability space of functions, which have uniform distribution and form $k$-wise independent random variables.

They can often be used in place of classic (or cryptographic) hash functions and allow the rigorous analysis of the performance of randomized algorithms and data structures that rely on hash functions.

In 1981 Wegman and Carter [4] introduced a generic construction for such families with arbitrary $k$ using polynomials over a finite field. This entry contains a formalization of them and establishes the property of $k$-universality.

To be useful the formalization also provides an explicit construction of finite fields using the factor ring of integers modulo a prime. Additionally, some generic results about independent families are shown that might be of independent interest.


## 1 Introduction and Definition

theory Universal-Hash-Families<br>imports $H O L-$ Probability.Independent-Family<br>begin

Universal hash families are commonly used in randomized algorithms and data structures to randomize the input of algorithms, such that probabilistic methods can be employed without requiring any assumptions about the input distribution.
If we regard a family of hash functions from a domain $D$ to a finite range $R$ as a uniform probability space, then the family is $k$-universal if:

- For each $x \in D$ the evaluation of the functions at $x$ forms a uniformly distributed random variable on $R$.
- The evaluation random variables for $k$ or fewer distinct domain elements form an independent family of random variables.

This definition closely follows the definition from Vadhan [3, §3.5.5], with the minor modification that independence is required not only for exactly $k$, but also for fewer than $k$ distinct domain elements. The correction is due to the fact that in the corner case where $D$ has fewer than $k$ elements, the second part of their definition becomes void. In the formalization this helps avoid an unnecessary assumption in the theorems.
The following definition introduces the notion of $k$-wise independent random variables:

```
definition (in prob-space) \(k\)-wise-indep-vars where
    \(k\)-wise-indep-vars \(k M^{\prime} X I=\)
        \(\left(\forall J \subseteq I\right.\). card \(J \leq k \longrightarrow\) finite \(J \longrightarrow\) indep-vars \(\left.M^{\prime} X J\right)\)
lemma (in prob-space) \(k\)-wise-indep-vars-subset:
    assumes \(k\)-wise-indep-vars \(k M^{\prime} X I\)
    assumes \(J \subseteq I\)
    assumes finite \(J\)
    assumes card \(J \leq k\)
    shows indep-vars \(M^{\prime} X J\)
    using assms
    by (simp add: \(k\)-wise-indep-vars-def)
lemma (in prob-space) \(k\)-wise-indep-subset:
    assumes \(J \subseteq I\)
    assumes \(k\)-wise-indep-vars \(k M^{\prime} X^{\prime} I\)
    shows \(k\)-wise-indep-vars \(k M^{\prime} X^{\prime} J\)
    using assms unfolding \(k\)-wise-indep-vars-def by simp
```

Similarly for a finite non-empty set $A$ the predicate uniform-on $X A$ indicates that the random variable is uniformly distributed on $A$ :

```
definition (in prob-space) uniform-on \(X A=\) (
    distr \(M\) (count-space UNIV) \(X=\) uniform-measure (count-space UNIV) \(A \wedge\)
    \(A \neq\{ \} \wedge\) finite \(A \wedge\) random-variable (count-space UNIV) X)
lemma (in prob-space) uniform-onD:
    assumes uniform-on \(X A\)
    shows prob \(\{\omega \in\) space \(M . X \omega \in B\}=\operatorname{card}(A \cap B) / \operatorname{card} A\)
proof -
    have \(\operatorname{prob}\{\omega \in\) space \(M . X \omega \in B\}=\operatorname{prob}\left(X-{ }^{\prime} B \cap\right.\) space \(\left.M\right)\)
        by (subst Int-commute, simp add:vimage-def Int-def)
    also have \(\ldots=\) measure (distr \(M\) (count-space UNIV) X) \(B\)
        using assms by (subst measure-distr, auto simp:uniform-on-def)
    also have \(\ldots=\) measure (uniform-measure (count-space UNIV) A) B
        using assms by (simp add:uniform-on-def)
    also have \(\ldots=\operatorname{card}(A \cap B) / \operatorname{card} A\)
        using assms by (subst measure-uniform-measure, auto simp:uniform-on-def)+
    finally show? ?thesis by simp
qed
```

With the two previous definitions it is possible to define the $k$-universality condition for a family of hash functions from $D$ to $R$ :

```
definition (in prob-space) \(k\)-universal \(k X D R=(\)
    \(k\)-wise-indep-vars \(k\) ( \(\lambda\)-. count-space UNIV) \(X D \wedge\)
    \((\forall i \in D\). uniform-on \((X i) R))\)
```

Note: The definition is slightly more generic then the informal specification from above. This is because usually a family is formed by a single function with a variable seed parameter. Instead of choosing a random function from a probability space, a random seed is chosen from the probability space which parameterizes the hash function.
The following section contains some preliminary results about independent families of random variables. Section 3 introduces the Carter-Wegman hash family, which is an explicit construction of $k$-universal families for arbitrary $k$ using polynomials over finite fields. The last section contains a proof that the factor ring of the integers modulo a prime ideal is a finite field, followed by an isomorphic construction of prime fields over an initial segment of the natural numbers.
end

## 2 Preliminary Results

```
theory Universal-Hash-Families-More-Independent-Families
    imports
        Universal-Hash-Families
        HOL-Probability.Stream-Space
        HOL-Probability.Probability-Mass-Function
begin
lemma set-comp-image-cong:
    assumes \(\bigwedge x . P x \Longrightarrow f x=h(g x)\)
    shows \(\{f x \mid x . P x\}=h '\{g x \mid x . P x\}\)
    using assms by (auto simp: setcompr-eq-image)
lemma (in prob-space) \(k\)-wise-indep-vars-compose:
    assumes \(k\)-wise-indep-vars \(k M^{\prime} X I\)
    assumes \(\bigwedge i . i \in I \Longrightarrow Y i \in\) measurable \(\left(M^{\prime} i\right)(N i)\)
    shows \(k\)-wise-indep-vars \(k N(\lambda i x . Y i(X i x)) I\)
    using indep-vars-compose2[where \(N=N\) and \(X=X\) and \(Y=Y\) and \(\left.M^{\prime}=M^{\prime}\right]\)
assms
    by (simp add: \(k\)-wise-indep-vars-def subsetD)
```

The following two lemmas are of independent interest, they help infer independence of events and random variables on distributions. (Candidates for HOL-Probability.Independent-Family).
lemma (in prob-space) indep-sets-distr:

```
    fixes }
    assumes random-variable Nf
    defines }F\equiv(\lambdai.(\lambdaa.f-' a\cap space M)'A i
    assumes indep-F: indep-sets FI
    assumes sets-A: \bigwedgei. i\inI\LongrightarrowA i\subseteq sets N
    shows prob-space.indep-sets (distr M Nf) A I
proof (rule prob-space.indep-setsI)
    show }\bigwedge\mp@subsup{A}{}{\prime}J.J\not={}\LongrightarrowJ\subseteqI\Longrightarrow finite J\Longrightarrow\forallj\inJ. A' j j A A j\Longrightarrow
        measure (distr M Nf) (\bigcap (A'`}J))=(\prodj\inJ. measure (distr M Nf) ( A' j)
    proof -
    fix }\mp@subsup{A}{}{\prime}
    assume a:J\subseteqI finite J J\not={}\forallj\inJ. A' j\inAj
    define F' where F' }\mp@subsup{F}{}{\prime}=(\lambdai.f-\mp@subsup{|}{}{\prime}\mp@subsup{A}{}{\prime}i\cap\mathrm{ space M)
    have \bigcap (F'`}J)=f-'(\bigcap(\mp@subsup{A}{}{\prime}`J))\cap\mathrm{ space M
        unfolding set-eq-iff F'-def using a(3) by simp
    moreover have }\bigcap(\mp@subsup{A}{}{\prime}'J)\in\mathrm{ sets N
        by (metis a sets-A sets.finite-INT subset-iff)
    ultimately have b:
        measure (distr M Nf) (\bigcap(A'`J)) = measure M (\bigcap(F'`J))
        by (metis assms(1) measure-distr)
    have }\j.j\inJ\Longrightarrow\mp@subsup{F}{}{\prime}j\inF
        using a(4) F''def F-def by blast
    hence c:measure M (\bigcap (F'`}J))=(\prodj\inJ. measure M (F'j)
        by (metis indep-F indep-setsD a(1,2,3))
    have }\bigwedgej.j\inJ\Longrightarrow\mp@subsup{F}{}{\prime}j=f-`\mp@subsup{A}{}{\prime}j\cap\mathrm{ space M
        by (simp add:F'-def)
    moreover have }\j.j\inJ\Longrightarrow\mp@subsup{A}{}{\prime}j\in\mathrm{ sets N
        using a(1,4) sets-A by blast
    ultimately have d
        \j.j\inJ\Longrightarrow measure M ( F'j j) = measure (distr M Nf) (A'j)
        using assms(1) measure-distr by metis
    show
        measure (distr M Nf) (\bigcap (A''J)) = (\prodj\inJ.measure (distr M Nf) ( (A'j))
        using b c d by auto
    qed
    show prob-space (distr MNf) using prob-space-distr assms by blast
    show \bigwedgei.i\inI\LongrightarrowA i\subseteq sets (distr M Nf) using sets-A sets-distr by blast
qed
lemma (in prob-space) indep-vars-distr:
    assumes f}\in\mathrm{ measurable MN
    assumes \i. i }\=|\mp@subsup{X}{}{\prime}i\in\mathrm{ measurable N(M'i)
    assumes indep-vars M'}\mp@subsup{M}{}{\prime}(\lambdai.(\mp@subsup{X}{}{\prime}i)\circf)
    shows prob-space.indep-vars (distr M Nf) M' X' I
```

```
proof -
    interpret D: prob-space (distr MNf)
        using prob-space-distr[OF assms(1)] by simp
    have a: f}\in\mathrm{ space M }M\mathrm{ space N using assms(1) by (simp add:measurable-def)
    have D.indep-sets (\lambdai. {X'i -' A\cap space N |A.A\in sets (M' i)})I
    proof (rule indep-sets-distr[OF assms(1)])
        have \i.i }\\\\Longrightarrow{(\mp@subsup{X}{}{\prime}i\circf)-'A\cap\mathrm{ space M |A.A sets ( }\mp@subsup{M}{}{\prime}i)}
```



```
            by (rule set-comp-image-cong, simp add:set-eq-iff, use a in blast)
        thus indep-sets (\lambdai. (\lambdaa.f -' a \cap space M)
                    {\mp@subsup{X}{}{\prime}i-'}A\cap\mathrm{ space N|A.A sets (M'i)})I
                using assms(3) by (simp add:indep-vars-def2 cong:indep-sets-cong)
    next
        fix }
        assume i\inI
        thus {X'i -' A\cap space N |A.A\in sets ( }\mp@subsup{M}{}{\prime}i)}\subseteq\mathrm{ sets N
            using assms(2) measurable-sets by blast
    qed
    thus ?thesis
        using assms by (simp add:D.indep-vars-def2)
qed
lemma range-inter: range ((\cap)F)=Pow F
    unfolding image-def by auto
```

The singletons and the empty set form an intersection stable generator of a countable discrete $\sigma$-algebra:

```
lemma sigma-sets-singletons-and-empty:
    assumes countable M
    shows sigma-sets M (insert {} ((\lambdak. {k})'M)) = Pow M
proof -
    have sigma-sets M ((\lambdak.{k})'M)= Pow M
        using assms sigma-sets-singletons by auto
    hence Pow M\subseteq sigma-sets M (insert {} (( }\lambdak.{k})`M)
        by (metis sigma-sets-subseteq subset-insertI)
    moreover have (insert {} ((\lambdak. {k})'M))\subseteq Pow M by blast
    hence sigma-sets M (insert {} ((\lambdak. {k})'M))\subseteq Pow M
        by (meson sigma-algebra.sigma-sets-subset sigma-algebra-Pow)
    ultimately show ?thesis by force
qed
```

In some of the following theorems, the premise $M=$ measure-pmf $p$ is used. This allows stating theorems that hold for pmfs more concisely, for example, instead of measure-pmf.prob $p A \leq$ measure-pmf.prob $p B$ we can just write $M=$ measure-pmf $p \Longrightarrow$ prob $A \leq$ prob $B$ in the locale prob-space.
lemma prob-space-restrict-space:
assumes $[$ simp $]: M=$ measure-pmf $p$

```
shows prob-space (restrict-space M (set-pmf p))
```

by (rule prob-spaceI, auto simp:emeasure-restrict-space emeasure-pmf)
The abbreviation below is used to specify the discrete $\sigma$-algebra on UNIV as a measure space. It is used in places where the existing definitions, such as indep-vars, expect a measure space even though only a measurable space is really needed, i.e., in cases where the property is invariant with respect to the actual measure.
hide-const (open) discrete
abbreviation discrete $\equiv$ count-space UNIV
lemma (in prob-space) indep-vars-restrict-space:
assumes $[$ simp $]: M=$ measure-pmf $p$
assumes
prob-space.indep-vars (restrict-space $M($ set-pmf $p))(\lambda$-. discrete) $X I$
shows indep-vars ( $\lambda$-. discrete) $X I$
proof -
have $a$ : id $\in$ restrict-space $M($ set-pmf $p) \rightarrow_{M} M$
by (simp add:measurable-def range-inter sets-restrict-space)
have prob-space.indep-vars (distr (restrict-space $M$ (set-pmf p)) Mid) ( $\lambda$-. discrete) $X I$
using assms a prob-space-restrict-space by (auto intro!:prob-space.indep-vars-distr) moreover have
$\bigwedge A$. emeasure (distr (restrict-space $M($ set-pmf p)) Mid) $A=$ emeasure $M A$ using emeasure-distr[OF a]
by (auto simp add: emeasure-restrict-space emeasure-Int-set-pmf)
hence distr (restrict-space Mp) Mid $=M$
by (auto intro: measure-eqI)
ultimately show? ?thesis by simp
qed
lemma (in prob-space) measure-pmf-eq:
assumes $M=$ measure- $p m f p$
assumes $\bigwedge x . x \in$ set-pmf $p \Longrightarrow(x \in P)=(x \in Q)$
shows prob $P=$ prob $Q$
unfolding assms(1)
by (rule measure-eq-AE, rule AE-pmfI[OF assms(2)], auto)
The following lemma is an intro rule for the independence of random variables defined on pmfs. In that case it is possible, to check the independence of random variables point-wise.
The proof relies on the fact that the support of a pmf is countable and the $\sigma$-algebra of such a set can be generated by singletons.
lemma (in prob-space) indep-vars-pmf:
assumes $[s i m p]: M=$ measure-pmf $p$
assumes $\bigwedge a J . J \subseteq I \Longrightarrow$ finite $J \Longrightarrow$
$\operatorname{prob}\{\omega . \forall i \in J . X i \omega=a i\}=\left(\prod i \in J . \operatorname{prob}\{\omega . X i \omega=a i\}\right)$
shows indep-vars ( $\lambda$-. discrete) $X I$

## proof -

interpret $R$ :prob-space (restrict-space $M$ (set-pmf p))
using prob-space-restrict-space by auto
have events-eq-pow: R.events $=$ Pow (set-pmf p)
by (simp add:sets-restrict-space range-inter)
define $G$ where $G=(\lambda i .\{\{ \}\} \cup(\lambda x .\{x\})$ ' $(X i$ ' set-pmf $p))$
define $F$ where $F=\left(\lambda i\right.$. $\left\{X i-{ }^{`} a \cap\right.$ set-pmf $\left.\left.p \mid a . a \in G i\right\}\right)$
have sigma-sets-pow:
$\bigwedge i . i \in I \Longrightarrow$ sigma-sets $(X i$ ' set-pmf $p)(G i)=\operatorname{Pow}(X i$ 'set-pmf $p)$
by (simp add:G-def, metis countable-image countable-set-pmf sigma-sets-singletons-and-empty)
have F-in-events: $\bigwedge i . i \in I \Longrightarrow F i \subseteq$ Pow (set-pmf $p$ )
unfolding $F$-def by blast
have as-sigma-sets:
$\bigwedge i . i \in I \Longrightarrow\left\{u . \exists A . u=X i-{ }^{\prime} A \cap\right.$ set-pmf $\left.p\right\}=$ sigma-sets $($ set-pmf $p)(F$
i)
proof -
fix $i$
assume $a: i \in I$
have $\bigwedge A . X i-{ }^{\prime} A \cap$ set-pmf $p=X i-‘(A \cap X i$ 'set-pmf $p) \cap$ set-pmf $p$ by auto
hence $\left\{u . \exists A . u=X i-{ }^{\prime} A \cap\right.$ set-pmf $\left.p\right\}=$
$\left\{X i-{ }^{\prime} A \cap\right.$ set-pmf $p \mid A . A \subseteq X i$ 'set-pmf $\left.p\right\}$ by (metis (no-types, opaque-lifting) inf-le2)
also have

$$
\ldots=\left\{X i-{ }^{`} A \cap \text { set-pmf } p \mid A . A \in \text { sigma-sets }(X i ' \text { set-pmf } p)(G i)\right\}
$$

using $a$ by (simp add:sigma-sets-pow)
also have $\ldots=$ sigma-sets (set-pmf $p)\left\{X i-{ }^{\prime} a \cap\right.$ set-pmf $\left.p \mid a . a \in G i\right\}$
by (subst sigma-sets-vimage-commute[symmetric], auto)
also have $\ldots=$ sigma-sets $($ set-pmf p) $(F i)$
by (simp add:F-def)
finally show
$\{u . \exists A . u=X i-‘ A \cap$ set-pmf $p\}=$ sigma-sets $($ set-pmf $p)(F i)$
by $\operatorname{simp}$
qed
have F-Int-stable: $\bigwedge i . i \in I \Longrightarrow$ Int-stable $(F i)$
proof (rule Int-stableI)
fix $i a b$
assume $i \in I \quad a \in F i \quad b \in F i$
thus $a \cap b \in(F i)$
unfolding $F$-def $G$-def by (cases $a \cap b=\{ \}$, auto)
qed

```
have F-indep-sets:R.indep-sets F I
proof (rule R.indep-setsI)
    fix i
    assume i\inI
    show Fi\subseteqR.events
        unfolding F-def events-eq-pow by blast
next
    fix }
    fix }
    assume a:J\subseteqIJ\not={} finite J\forallj\inJ. Aj\inFj
    have b: \bigwedgej.j\inJ\LongrightarrowAj\subseteq set-pmf p
    by (metis PowD a(1,4) subsetD F-in-events)
    obtain x where x-def:\j.j 
    using a by (simp add:Pi-def F-def, metis)
    show R.prob (\bigcap (A`J)) = (\prodj\inJ. R.prob (A j))
    proof (cases \existsj\inJ.Aj={})
        case True
        hence }\bigcap(A'J)={} by blas
        then show ?thesis
            using a True by (simp, metis measure-empty)
    next
        case False
        then have }\j.j\inJ\Longrightarrowxj\not={}\mathrm{ using x-def by auto
        hence }\j.j\inJ\Longrightarrowxj\in(\lambdax.{x})'Xj'set-pmf 
            using x-def by (simp add:G-def)
    then obtain y where y-def: \j.j\inJ\Longrightarrowxj={yj}
        by (simp add:image-def, metis)
    have }\cap(A'J)\subseteq\mathrm{ set-pmf p using b a(2) by blast
    hence R.prob (\bigcap (A'J)) = prob (\bigcapj\inJ.A j)
        by (simp add: measure-restrict-space)
    also have ... = prob ({\omega.\forallj\inJ.Xj\omega=yj})
        using a x-def y-def apply (simp add:vimage-def measure-Int-set-pmf)
        by (rule arg-cong2 [where f=measure], auto)
    also have .. = (\prod j\in J. prob (A j))
        using x-def y-def a assms(2)
        by (simp add:vimage-def measure-Int-set-pmf)
    also have ... = (\prodj\inJ. R.prob (Aj))
            using b by (simp add: measure-restrict-space cong:prod.cong)
    finally show ?thesis by blast
    qed
qed
have R.indep-sets (\lambdai. sigma-sets (set-pmf p) (Fi))I
    using R.indep-sets-sigma[simplified] F-Int-stable F-indep-sets
    by (auto simp:space-restrict-space)
```

```
    hence R.indep-sets (\lambdai. {u. \existsA.u=Xi-` A\cap set-pmf p})I
    by (simp add:as-sigma-sets cong:R.indep-sets-cong)
    hence R.indep-vars ( }\lambda\mathrm{ -. discrete) X I
    unfolding R.indep-vars-def2
    by (simp add:measurable-def sets-restrict-space range-inter)
    thus ?thesis
    using indep-vars-restrict-space[OF assms(1)] by simp
qed
lemma (in prob-space) split-indep-events:
    assumes }M=\mathrm{ measure-pmf p
    assumes indep-vars (\lambdai. discrete) X'I
    assumes K\subseteqI finite K
    shows prob {\omega.\forallx\inK.Px (\mp@subsup{X}{}{\prime}x\omega)}=(\prodx\inK.prob {\omega. Px ( (X'x\omega)})
proof -
    have [simp]: space M = UNIV events = UNIV prob UNIV = 1
        by (simp add:assms(1))+
    have indep-vars ( }\lambda\mathrm{ -. discrete) }\mp@subsup{X}{}{\prime}
    using assms(2,3) indep-vars-subset by blast
    hence indep-events ( }\lambdax.{\omega\in\mathrm{ space M. P x ( ( ' x }\omega\mathrm{ ) })K
    using indep-eventsI-indep-vars by force
    hence a:indep-events ( }\lambdax.{\omega.Px(\mp@subsup{X}{}{\prime}x\omega)})
    by simp
    have prob {\omega.}\forallx\inK.Px(\mp@subsup{X}{}{\prime}x\omega)}=\operatorname{prob}(\bigcapx\inK.{\omega.Px(和x\omega)}
    by (simp add: measure-pmf-eq[OF assms(1)])
    also have ... = (\Pix\inK. prob {\omega. Px ( ( ''x\omega)})
    using a assms(4) by (cases K={}, auto simp: indep-events-def)
    finally show ?thesis by simp
qed
lemma pmf-of-set-eq-uniform:
    assumes finite A A}={
    shows measure-pmf (pmf-of-set A) = uniform-measure discrete A
proof -
    have a:real (card A) > 0 using assms
        by (simp add: card-gt-0-iff)
    have b:
        \Y. emeasure (pmf-of-set A) Y= emeasure (uniform-measure discrete A) Y
        using assms a
        by (simp add: emeasure-pmf-of-set divide-ennreal ennreal-of-nat-eq-real-of-nat)
    show ?thesis
    by (rule measure-eqI, auto simp add: b)
qed
```

```
lemma (in prob-space) uniform-onI:
    assumes }M=\mathrm{ measure-pmf p
    assumes finite A A}={
    assumes \a.prob {\omega.X \omega=a}= indicator A a/card A
    shows uniform-on X A
proof -
    have a:\a. measure-pmf.prob p {x. X x =a}= indicator A a / card A
        using assms(1,4) by simp
    have b:map-pmf X p = pmf-of-set A
        by (rule pmf-eqI, simp add:assms pmf-map vimage-def a)
    have distr M discrete X = map-pmf X p
        by (simp add: map-pmf-rep-eq assms(1))
    also have ... = measure-pmf (pmf-of-set A)
        using b by simp
    also have ... = uniform-measure discrete A
        by (rule pmf-of-set-eq-uniform[OF assms(2,3)])
    finally have distr M discrete X = uniform-measure discrete A
        by simp
    moreover have random-variable discrete X
        by (simp add: assms(1))
    ultimately show ?thesis using assms(2,3)
        by (simp add: uniform-on-def)
qed
end
```


## 3 Carter-Wegman Hash Family

theory Carter-Wegman-Hash-Family<br>imports

Interpolation-Polynomials-HOL-Algebra.Interpolation-Polynomial-Cardinalities Universal-Hash-Families-More-Independent-Families
begin
The Carter-Wegman hash family is a generic method to obtain $k$-universal hash families for arbitrary $k$. (There are faster solutions, such as tabulation hashing, which are limited to a specific $k$. See for example [2].)
The construction was described by Wegman and Carter [4], it is a hash family between the elements of a finite field and works by choosing randomly a polynomial over the field with degree less than $k$. The hash function is the evaluation of a such a polynomial.
Using the property that the fraction of polynomials interpolating a given set of $s \leq k$ points is $1 /$ real $(\operatorname{card}(\text { carrier } R))^{s}$, which is shown in [1], it is possible to obtain both that the hash functions are $k$-wise independent and
uniformly distributed.
In the following two locales are introduced, the main reason for both is to make the statements of the theorems and proofs more concise. The first locale poly-hash-family fixes a finite ring $R$ and the probability space of the polynomials of degree less than $k$. Because the ring is not a field, the family is not yet $k$-universal, but it is still possible to state a few results such as the fact that the range of the hash function is a subset of the carrier of the ring.
The second locale carter-wegman-hash-family is an extension of the former with the assumption that $R$ is a field with which the $k$-universality follows.
The reason for using two separate locales is to support use cases, where the ring is only probably a field. For example if it is the set of integers modulo an approximate prime, in such a situation a subset of the properties of an algorithm using approximate primes would need to be verified even if $R$ is only a ring.

```
definition (in ring) hash \(x \omega=\operatorname{eval} \omega x\)
locale poly-hash-family \(=\) ring +
    fixes \(k\) :: nat
    assumes finite-carrier [simp]: finite (carrier \(R\) )
    assumes \(k\)-ge- \(0: k>0\)
begin
```

definition space where space $=$ bounded-degree-polynomials $R k$
definition $M$ where $M=$ measure-pmf (pmf-of-set space)
lemma finite-space[simp]:finite space
unfolding space-def using fin-degree-bounded finite-carrier by simp
lemma non-empty-bounded-degree-polynomials[simp]:space $\neq\{ \}$
unfolding space-def using non-empty-bounded-degree-polynomials by simp

This is to add carrier-not-empty to the simp set in the context of poly-hash-family:

```
lemma non-empty-carrier[simp]: carrier \(R \neq\{ \}\)
    by (simp add:carrier-not-empty)
sublocale prob-space \(M\)
    by (simp add:M-def prob-space-measure-pmf)
lemma hash-range[simp]:
    assumes \(\omega \in\) space
    assumes \(x \in\) carrier \(R\)
    shows hash \(x \omega \in\) carrier \(R\)
    using assms unfolding hash-def space-def bounded-degree-polynomials-def
    by (simp, metis eval-in-carrier polynomial-incl univ-poly-carrier)
```

```
lemma hash-range-2:
    assumes }\omega\in\mathrm{ space
    shows (\lambdax. hash x \omega)' carrier R}\subseteq\mathrm{ carrier R
    using hash-range assms by auto
lemma integrable-M[simp]:
    fixes f :: 'a list }=>\mp@subsup{}{}{\prime}c::{\mathrm{ banach, second-countable-topology}
    shows integrable Mf
        unfolding M-def
        by (rule integrable-measure-pmf-finite, simp)
end
locale carter-wegman-hash-family = poly-hash-family +
    assumes field-R: field R
begin
sublocale field
    using field-R by simp
abbreviation field-size }\equiv\operatorname{card}(\mathrm{ carrier R)
lemma poly-cards:
    assumes K\subseteqcarrier R
    assumes card K\leqk
    assumes y'}K\subseteq(carrier R
    shows
        card {\omega\in space. (\forallk\inK. eval }\omegak=yk)}=\mathrm{ field-size^(k-card K)
    unfolding space-def
    using interpolating-polynomials-card[where n=k-card K and K=K] assms
    using finite-carrier finite-subset by fastforce
lemma poly-cards-single:
    assumes x carrier R
    assumes y f carrier R
    shows card {\omega\in space. eval }\omegax=y}=field-size` (k-1
    using poly-cards[where K}={x}\mathrm{ and }y=\lambda-.y\mathrm{ , simplified] assms k-ge-0 by simp
lemma hash-prob:
    assumes K\subseteqcarrier R
    assumes card K
    assumes y'}K\subseteq\mathrm{ carrier R
    shows
    prob {\omega. (\forallx\inK. hash x \omega = y x)}=1/(real field-size)^card K
proof -
    have 0}\in\mathrm{ carrier R by simp
    hence a:field-size > 0
    using finite-carrier card-gt-0-iff by blast
```

```
    have b:real (card {\omega\in space. }\forallx\inK. eval \omegax=y x}) / real (card space) =
        1 / real field-size ^ card K
        using a assms(2)
    apply (simp add: frac-eq-eq poly-cards[OF assms(1,2,3)] power-add[symmetric])
    by (simp add:space-def bounded-degree-polynomials-card)
    show ?thesis
        unfolding M-def
        by (simp add:hash-def measure-pmf-of-set Int-def b)
qed
lemma prob-single:
    assumes x carrier R y c carrier R
    shows prob {\omega. hash x \omega=y}=1/(real field-size)
    using hash-prob[where K={x}] assms finite-carrier k-ge-0 by simp
lemma prob-range:
    assumes [simp]:x\in carrier R
    shows prob {\omega. hash x \omega\inA}=card (A\cap carrier R)/ field-size
proof -
    have prob {\omega. hash x \omega\inA} = prob ( \bigcupa\inA\cap carrier R. {\omega. hash x }\omega=a}
        by (rule measure-pmf-eq, auto simp:M-def)
    also have ... = (\suma\in(A\capcarrier R). prob {\omega. hash x \omega=a})
    by (rule measure-finite-Union, auto simp:M-def disjoint-family-on-def)
    also have ... = (\suma\in(A\cap carrier R). 1/(real field-size )}
    by (rule sum.cong, auto simp:prob-single)
    also have ... = card ( }A\cap\mathrm{ carrier R) / field-size
        by simp
    finally show ?thesis by simp
qed
lemma indep:
    assumes }J\subseteq\mathrm{ carrier R
    assumes card J\leqk
    shows indep-vars (\lambda-. discrete) hash J
proof -
    have 0}\in\mathrm{ carrier R by simp
    hence card-R-ge-0:field-size > 0
        using card-gt-0-iff finite-carrier by blast
    have fin-J: finite }
    using finite-carrier assms(1) finite-subset by blast
    show ?thesis
    proof (rule indep-vars-pmf[OF M-def])
        fix a
        fix }\mp@subsup{J}{}{\prime
        assume a: J'\subseteqJ finite }\mp@subsup{J}{}{\prime
    have card- -J':card J'}\leq
```

```
    by (metis card-mono order-trans a(1) assms(2) fin-J)
    have \(J^{\prime}\)-in-carr: \(J^{\prime} \subseteq\) carrier \(R\) by (metis order-trans a(1) assms(1))
    show prob \(\left\{\omega . \forall x \in J^{\prime}\right.\). hash \(\left.x \omega=a x\right\}=\left(\prod x \in J^{\prime} . \operatorname{prob} \quad\{\omega\right.\). hash \(\left.x \omega=a x\}\right)\)
    proof (cases a ' \(J^{\prime} \subseteq\) carrier \(R\) )
        case True
        have \(a\)-carr: \(\bigwedge x . x \in J^{\prime} \Longrightarrow a x \in\) carrier \(R\) using True by force
    have prob \(\left\{\omega\right.\). \(\forall x \in J^{\prime}\). hash \(\left.x \omega=a x\right\}=\)
        real (card \(\left\{\omega \in\right.\) space. \(\forall x \in J^{\prime}\). eval \(\left.\omega x=a x\right\}\) ) / real (card space)
        by (simp add:M-def measure-pmf-of-set Int-def hash-def)
    also have \(\ldots=\) real (field-size ^ \(\left(k-\right.\) card \(\left.\left.J^{\prime}\right)\right) /\) real (card space)
        using True by (simp add: poly-cards[OF J'-in-carr card- \(\left.J^{\prime}\right]\) )
    also have
        \(\ldots\) = real field-size ^ \(\left(k-\right.\) card \(\left.J^{\prime}\right) /\) real field-size \({ }^{\wedge} k\)
        by (simp add:space-def bounded-degree-polynomials-card)
    also have
        \(\ldots=\) real field-size ^\(\left((k-1) *\right.\) card \(\left.J^{\prime}\right) /\) real field-size ^ \(\left(k *\right.\) card \(\left.J^{\prime}\right)\)
        using card- \(J^{\prime}\) by (simp add:power-add[symmetric] power-mult[symmetric]
                diff-mult-distrib frac-eq-eq add.commute)
    also have
        \(\ldots=(\text { real field-size ^}(k-1))^{\wedge}\) card \(J^{\prime} /(\) real field-size \(\wedge k) \wedge\) card \(J^{\prime}\)
        by (simp add:power-add power-mult)
    also have
        \(\ldots=\left(\prod x \in J^{\prime}\right.\). real (card \(\{\omega \in\) space. eval \(\left.\omega x=a x\}\right) /\) real (card space) \()\)
        using a-carr poly-cards-single[OF subsetD[OF \(J^{\prime}\)-in-carr]]
        by (simp add:space-def bounded-degree-polynomials-card power-divide)
    also have \(\ldots=\left(\prod x \in J^{\prime} . \operatorname{prob}\{\omega\right.\). hash \(\left.x \omega=a x\}\right)\)
        by (simp add:measure-pmf-of-set M-def Int-def hash-def)
    finally show ?thesis by simp
    next
    case False
    then obtain \(j\) where \(j\)-def: \(j \in J^{\prime}\) a \(j \notin\) carrier \(R\) by blast
    have \(\{\omega \in\) space. hash \(j \omega=a j\} \subseteq\{\omega \in\) space. hash \(j \omega \notin\) carrier \(R\}\)
        by (rule subsetI, simp add:j-def)
    also have \(\ldots \subseteq\left\}\right.\) using \(j\)-def(1) \(J^{\prime}\)-in-carr hash-range by blast
    finally have \(b:\{\omega \in\) space. hash \(j \omega=a j\}=\{ \}\) by simp
    hence real (card \((\{\omega \in\) space. hash \(j \omega=a j\}))=0\) by simp
    hence \(\left(\prod x \in J^{\prime}\right.\). real \((\) card \(\{\omega \in\) space. hash \(\left.x \omega=a x\})\right)=0\)
        using \(a(2)\) prod-zero[OF \(a(2)] j\)-def(1) by auto
    moreover have
        \(\left\{\omega \in\right.\) space. \(\forall x \in J^{\prime}\). hash \(\left.x \omega=a x\right\} \subseteq\{\omega \in\) space. hash \(j \omega=a j\}\)
        using \(j\)-def by blast
    hence \(\left\{\omega \in\right.\) space. \(\forall x \in J^{\prime}\). hash \(\left.x \omega=a x\right\}=\{ \}\) using \(b\) by blast
    ultimately show ?thesis
        by (simp add:measure-pmf-of-set \(M\)-def Int-def prod-dividef)
        qed
    qed
qed
```

```
lemma k-wise-indep
    k-wise-indep-vars k ( }\lambda\mathrm{ -. discrete) hash (carrier R)
    unfolding }k\mathrm{ -wise-indep-vars-def using indep by simp
lemma inj-if-degree-1:
    assumes }\omega\in\mathrm{ space
    assumes degree \omega=1
    shows inj-on ( }\lambdax\mathrm{ . hash x }\omega\mathrm{ ) (carrier R)
    using assms eval-inj-if-degree-1
    by (simp add:M-def space-def bounded-degree-polynomials-def hash-def)
lemma uniform:
    assumes i\in carrier R
    shows uniform-on (hash i) (carrier R)
proof -
    have a
        \a.prob {\omega. hash i\omega\in{a}}= indicat-real (carrier R) a / real field-size
        by (subst prob-range[OF assms], simp add:indicator-def)
    show ?thesis
    by (rule uniform-onI, use a M-def in auto)
qed
This the main result of this section - the Carter-Wegman hash family is \(k\)-universal.
theorem \(k\)-universal:
\(k\)-universal \(k\) hash (carrier \(R\) ) (carrier \(R\) )
using uniform \(k\)-wise-indep by (simp add:k-universal-def)
end
lemma poly-hash-familyI:
assumes ring \(R\)
assumes finite (carrier \(R\) )
assumes \(0<k\)
shows poly-hash-family \(R k\)
using assms
by (simp add:poly-hash-family-def poly-hash-family-axioms-def)
lemma carter-wegman-hash-familyI:
assumes field \(F\)
assumes finite (carrier F)
assumes \(0<k\)
shows carter-wegman-hash-family F \(k\)
using assms field.is-ring[OF assms(1)] poly-hash-familyI
by (simp add:carter-wegman-hash-family-def carter-wegman-hash-family-axioms-def)
lemma hash-k-wise-indep:
assumes field \(F \wedge\) finite (carrier \(F\) )
assumes \(1 \leq n\)
```

```
shows
    prob-space.k-wise-indep-vars (pmf-of-set (bounded-degree-polynomials F n)) n
    (\lambda-. pmf-of-set (carrier F)) (ring.hash F) (carrier F)
proof -
    interpret carter-wegman-hash-family F n
        using assms carter-wegman-hash-familyI by force
    have k-wise-indep-vars n (\lambda-. pmf-of-set (carrier F)) hash (carrier F)
        by (rule k-wise-indep-vars-compose[OF k-wise-indep], simp)
    thus ?thesis
        by (simp add:M-def space-def)
qed
lemma hash-prob-single:
    assumes field F}\wedge finite (carrier F
    assumes }x\in\mathrm{ carrier F
    assumes 1\leqn
    assumes y f carrier F
    shows
        P}(\omega\mathrm{ in pmf-of-set (bounded-degree-polynomials F n). ring.hash F x w = y)
        =1/(real (card (carrier F)))
proof -
    interpret carter-wegman-hash-family F n
        using assms carter-wegman-hash-familyI by force
    show ?thesis
        using prob-single[OF assms(2,4)] by (simp add:M-def space-def)
qed
end
```


## 4 Finite Fields

```
theory Universal-Hash-Families-More-Finite-Fields
    imports Finite-Fields.Finite-Fields-Mod-Ring-Code
begin
```

This theory have been moved to Finite-Fields.Finite-Fields-Mod-Ring-Code, where mod-ring $n$ corresponds to ring-of (mod-ring $n$ ). The lemmas and definitions here are kept to prevent merge-conflicts.
lemmas zfact-iso-0 $=$ zfact-iso-0
lemmas zfact-prime-is-field $=$ zfact-prime-is-field
hide-const (open) Multiset.mult
definition mod-ring :: nat => nat ring
where mod-ring $n=0$

```
carrier = {..<n},
    mult = (\lambdaxy.(x*y)mod n),
    one =1,
```

```
    zero = 0,
    add = (\lambda x y. (x+y) mod n) )
definition zfact-iso-inv :: nat }=>\mathrm{ int set }=>\mathrm{ nat where
    zfact-iso-inv p = inv-into {..<p}(zfact-iso p)
lemma zfact-iso-inv-compat:
    assumes n>0
    assumes }x\in\operatorname{carrier (ZFact (int n))
    shows zfact-iso-inv n x = Finite-Fields-Mod-Ring-Code.zfact-iso-inv n x
proof -
    have Finite-Fields-Mod-Ring-Code.zfact-iso-inv n x \in{..<n}
        using bij-betw-apply[OF zfact-iso-inv-bij[OF assms(1)] assms(2)]
        by (simp add:Finite-Fields-Mod-Ring-Code.mod-ring-def ring-of-def)
    moreover have zfact-iso n (Finite-Fields-Mod-Ring-Code.zfact-iso-inv n x) =x
        unfolding Finite-Fields-Mod-Ring-Code.zfact-iso-inv-def
        using zfact-iso-bij[OF assms(1)] assms(2)
        by (intro f-the-inv-into-f) (simp-all add:bij-betw-def)
    ultimately show ?thesis
        unfolding zfact-iso-inv-def by (intro inv-into-f-eq zfact-iso-inj assms) auto
qed
lemma mod-ring-compat:
    mod-ring n = ring-of (Finite-Fields-Mod-Ring-Code.mod-ring n)
    unfolding mod-ring-def Finite-Fields-Mod-Ring-Code.mod-ring-def ring-of-def
by auto
lemma zfact-iso-inv-0:
    assumes n-ge-0: n > 0
    shows zfact-iso-inv n 0}\mp@subsup{\mathbf{0}}{\mathrm{ ZFact (int n)}}{}=0(\mathrm{ is ?L = ?R)
proof -
    interpret r:cring (ZFact (int n)) using ZFact-is-cring by simp
    have ?L = Finite-Fields-Mod-Ring-Code.zfact-iso-inv n 0}\mp@subsup{\mathbf{0}}{\mathrm{ ZFact (int n)}}{
    by (intro zfact-iso-inv-compat[OF assms]) simp
    also have ... = 0 using zfact-iso-inv-0[OF assms] by simp
    finally show ?thesis by simp
qed
lemma zfact-coset:
    assumes n-ge-0: n>0
    assumes x carrier (ZFact (int n))
    defines }I\equivId\mp@subsup{l}{\mathcal{Z}}{{}{\mathrm{ int n}
    shows }x=I+\mp@subsup{>}{\mathcal{Z}}{}(\mathrm{ int (zfact-iso-inv n }n\mathrm{ ) )
proof -
    have x=I +>>\mathcal{Z }}\mathrm{ (int (Finite-Fields-Mod-Ring-Code.zfact-iso-inv n x))
        unfolding I-def by (intro zfact-coset[OF assms(1,2)])
    also have ... = I +> \mathcal{Z }}\mathrm{ (int (zfact-iso-inv n x))
    using zfact-iso-inv-compat[OF assms(1,2)] by simp
```

```
    finally show ?thesis by simp
qed
lemma zfact-iso-inv-is-ring-iso:
    assumes n-ge-1:n>1
    shows zfact-iso-inv n \in ring-iso (ZFact (int n)) (mod-ring n)
proof -
    interpret r:cring (ZFact (int n)) using ZFact-is-cring by simp
    show ?thesis
    unfolding mod-ring-compat using assms
    by (intro r.ring-iso-restrict[OF zfact-iso-inv-is-ring-iso[OF n-ge-1]]
            zfact-iso-inv-compat[symmetric]) auto
qed
lemma mod-ring-finite:
    finite (carrier (mod-ring n))
    using mod-ring-finite mod-ring-compat by auto
lemma mod-ring-carr:
    x\in carrier (mod-ring n) \longleftrightarrow x<n
    using mod-ring-carr mod-ring-compat by auto
lemma mod-ring-is-cring:
    assumes n-ge-1: n> 1
    shows cring (mod-ring n)
    using mod-ring-is-cring[OF assms] mod-ring-compat by auto
lemma zfact-iso-is-ring-iso:
    assumes n-ge-1:n> 1
    shows zfact-iso n \in ring-iso (mod-ring n) (ZFact (int n))
    using zfact-iso-is-ring-iso[OF assms] mod-ring-compat by auto
```

If $p$ is a prime than Universal-Hash-Families-More-Finite-Fields.mod-ring p
is a field:
lemma mod-ring-is-field:
assumesFactorial-Ring.prime $p$
shows field (mod-ring $p$ )
using mod-ring-is-field[OF assms] mod-ring-compat by auto
end

## 5 Indexed Products of Probability Mass Functions

```
theory Universal-Hash-Families-More-Product-PMF
    imports
        HOL-Probability.Product-PMF
        Concentration-Inequalities.Concentration-Inequalities-Preliminary
        Finite-Fields.Finite-Fields-More-Bijections
```

hide-const (open) Isolated.discrete
This section introduces a restricted version of Pi-pmf where the default value is undefined and contains some additional results about that case in addition to HOL-Probability.Product-PMF
abbreviation prod-pmf where prod-pmf I M $\equiv$ Pi-pmf I undefined $M$
lemma measure-pmf-cong:
assumes $\backslash x . x \in$ set-pmf $p \Longrightarrow x \in P \longleftrightarrow x \in Q$
shows measure (measure-pmf p) $P=$ measure (measure-pmf p) $Q$
using assms
by (intro finite-measure.finite-measure-eq-AE AE-pmfI) auto
lemma pmf-mono:
assumes $\bigwedge x . x \in$ set-pmf $p \Longrightarrow x \in P \Longrightarrow x \in Q$
shows measure (measure-pmf p) $P \leq$ measure (measure-pmf p) $Q$
proof -
have measure (measure-pmf $p$ ) $P=$ measure (measure-pmf $p)(P \cap($ set-pmf $p))$
by (intro measure-pmf-cong) auto
also have $\ldots \leq$ measure ( measure-pmf p) $Q$
using assms by (intro finite-measure.finite-measure-mono) auto
finally show?thesis by simp
qed
lemma $p m f-a d d$ :
assumes $\backslash x . x \in P \Longrightarrow x \in \operatorname{set}-p m f p \Longrightarrow x \in Q \vee x \in R$
shows measure $p P \leq$ measure $p Q+$ measure $p R$
proof -
have measure $p P \leq$ measure $p(Q \cup R)$
using assms by (intro pmf-mono) blast
also have $\ldots \leq$ measure $p Q+$ measure $p R$
by (rule measure-subadditive, auto)
finally show?thesis by simp
qed
lemma pmf-prod-pmf:
assumes finite I
shows pmf (prod-pmf IM)x=(if $x \in$ extensional I then $\prod i \in I .(p m f(M i))$
( $x$ i) else 0)
by (simp add: pmf-Pi[OF assms(1)] extensional-def)
lemma PiE-defaut-undefined-eq: PiE-dflt I undefined $M=$ PiE I M
by (simp add:PiE-dftt-def PiE-def extensional-def Pi-def set-eq-iff) blast
lemma set-prod-pmf:
assumes finite $I$

```
shows set-pmf (prod-pmf I M) = PiE I (set-pmf ○M)
```

by (simp add:set-Pi-pmf[OF assms] PiE-defaut-undefined-eq)

A more general version of measure-Pi-pmf-Pi

```
lemma prob-prod-pmf \({ }^{\prime}\) :
    assumes finite \(I\)
    assumes \(J \subseteq I\)
    shows measure (measure-pmf (Pi-pmf I d M) ) (Pi J A) \(=\left(\prod i \in J\right.\). measure
(Mi) (Ai))
proof -
    have \(a: P i J A=P i I(\lambda i\). if \(i \in J\) then \(A\) i else UNIV)
    using assms by (simp add:Pi-def set-eq-iff, blast)
    show ?thesis
    using assms
        by (simp add:if-distrib a measure-Pi-pmf-Pi[OF assms(1)] prod.If-cases[OF
\(\operatorname{assms}(1)]\) Int-absorb1)
qed
lemma prob-prod-pmf-slice:
    assumes finite I
    assumes \(i \in I\)
    shows measure (measure-pmf (prod-pmf IM)) \(\{\omega . P(\omega i)\}=\) measure \(\left(\begin{array}{ll}( & i\end{array}\right)\)
\(\{\omega . P \omega\}\)
    using prob-prod-pmf \({ }^{\prime}[O F \operatorname{assms}(1)\), where \(J=\{i\}\) and \(M=M\) and \(A=\lambda\)-. Col-
lect \(P\) ]
    by (simp add:assms Pi-def)
```

definition restrict-dfl where restrict-dfl $f A d=(\lambda x$. if $x \in A$ then $f x$ else $d)$
lemma pi-pmf-decompose:
assumes finite I
shows Pi-pmf I d $M=$ map-pmf $(\lambda \omega$. restrict-dfl $(\lambda i . \omega(f i) i) I d)(P i-p m f(f$
$\left.\left.{ }^{\prime} I\right)(\lambda-. d)(\lambda j . \operatorname{Pi}-p m f(f-‘\{j\} \cap I) d M)\right)$
proof -
have fin-F-I:finite ( $f$ ' $I$ ) using assms by blast
have finite $I \Longrightarrow$ ?thesis
using fin-F-I
proof (induction f'I arbitrary: I rule:finite-induct)
case empty
then show ?case by (simp add:restrict-dfl-def)
next
case (insert $x F$ )
have $a:(f-‘\{x\} \cap I) \cup(f-‘ F \cap I)=I$
using insert(4) by blast
have $b: F=f^{\prime}(f-‘ F \cap I)$ using $\operatorname{insert}(2,4)$
by (auto simp add:set-eq-iff image-def vimage-def)
have $c$ : finite $\left(f-{ }^{\prime} F \cap I\right)$ using insert by blast
have $d: \bigwedge j . j \in F \Longrightarrow(f-‘\{j\} \cap(f-' F \cap I))=(f-‘\{j\} \cap I)$
using insert(4) by blast
have Pi-pmf IdM=Pi-pmf $((f-‘\{x\} \cap I) \cup(f-‘ F \cap I)) d M$ by ( $\operatorname{simp}$ add:a)
also have $\ldots=\operatorname{map-pmf}(\lambda(g, h)$ i. if $i \in f-‘\{x\} \cap I$ then $g$ i else $h i)$ (pair-pmf $\left.(\operatorname{Pi-pmf}(f-‘\{x\} \cap I) d M)\left(\operatorname{Pi-pmf}\left(f-{ }^{\prime} F \cap I\right) d M\right)\right)$ using insert by (subst Pi-pmf-union) auto
also have $\ldots=\operatorname{map-pmf}(\lambda(g, h)$. if $f i=x \wedge i \in I$ then $g$ i else iff $i \in F \wedge$ $i \in I$ then $h(f i) i$ else $d)$ (pair-pmf $(\operatorname{Pi-pmf}(f-‘\{x\} \cap I) d M)(\operatorname{Pi-pmf} F(\lambda-. d)(\lambda j . \operatorname{Pi}-p m f(f-‘$ $\{j\} \cap(f-‘ F \cap I)) d M)))$ by (simp add:insert(3)[ $\left.\begin{array}{lll}O F & b & c\end{array}\right]$ map-pmf-comp case-prod-beta' apsnd-def map-prod-def
pair-map-pmf2 b[symmetric] restrict-dfl-def) (metis fst-conv snd-conv)
also have $\ldots=\operatorname{map-pmf}(\lambda(g, h)$ i. if $i \in I$ then $(h(x:=g))(f i) i$ else $d)$ (pair-pmf $(\operatorname{Pi-pmf}(f-‘\{x\} \cap I) d M)(\operatorname{Pi-pmf} F(\lambda-. d)(\lambda j$. Pi-pmf $(f-‘$ $\{j\} \cap I) d M))$ ) using insert(4) d
by (intro arg-cong2[where $f=$ map-pmf] ext) (auto simp add:case-prod-beta' cong:Pi-pmf-cong)
also have $\ldots=$ map-pmf $(\lambda \omega$. if $i \in I$ then $\omega(f i)$ i else d) (Pi-pmf (insert $x F)(\lambda-. d)(\lambda j$. Pi-pmf $(f-‘\{j\} \cap I) d M))$
by (simp add:Pi-pmf-insert[OF insert(1, 2)] map-pmf-comp case-prod-beta')
finally show? ?ase by (simp add:insert(4) restrict-dfl-def)
qed
thus ?thesis using assms by blast
qed
lemma restrict-dfl-iter: restrict-dfl (restrict-dflfId) Jd=restrict-dfl $f(I \cap J)$ d
by (rule ext, simp add:restrict-dfl-def)
lemma indep-vars-restrict ${ }^{\prime}$ :
assumes finite I
shows prob-space.indep-vars (Pi-pmf I d M) ( $\lambda$-. discrete) ( $\lambda i \omega$. restrict-dfl $\omega$ $(f-‘\{i\} \cap I) d)\left(f^{\prime} I\right)$
proof -
let ? $Q=(\operatorname{Pi-pmf}(f ‘ I)(\lambda-. d)(\lambda i . \operatorname{Pi-pmf}(I \cap f-‘\{i\}) d M))$
have a:prob-space.indep-vars? $Q$ ( $\lambda$-. discrete) $(\lambda i \omega . \omega i)\left(f^{\prime} I\right)$
using assms by (intro indep-vars-Pi-pmf, blast)
have b: AE $x$ in measure-pmf ?Q. $\forall i \in f$ ' $I . x i=$ restrict-dfl $(\lambda i . x(f i) i)(I \cap$ $f-‘\{i\}) d$
using assms
by (auto simp add:PiE-dflt-def restrict-dfl-def AE-measure-pmf-iff set-Pi-pmf comp-def Int-commute)
have prob-space.indep-vars ?Q ( $\lambda$-. discrete) ( $\lambda i$ x. restrict-dfl ( $\lambda i . x(f i)$ i) $(I$ $\cap f-‘\{i\}) d)\left(f^{\prime} I\right)$
by (rule prob-space.indep-vars-cong-AE[OF prob-space-measure-pmf ba], simp) thus ?thesis
using prob-space-measure-pmf
by (auto intro!:prob-space.indep-vars-distr simp:pi-pmf-decompose[OF assms, where $f=f$ ]
map-pmf-rep-eq comp-def restrict-dfl-iter Int-commute)
qed
lemma indep-vars-restrict-intro ${ }^{\prime}$ :
assumes finite $I$
assumes $\bigwedge i \omega . i \in J \Longrightarrow X^{\prime} i \omega=X^{\prime} i($ restrict-dfl $\omega(f-‘\{i\} \cap I) d)$
assumes $J=f$ 'I
assumes $\bigwedge \omega i . i \in J \Longrightarrow X^{\prime} i \omega \in \operatorname{space}\left(M^{\prime} i\right)$
shows prob-space.indep-vars (measure-pmf (Pi-pmf Idp)) M' $\left(\lambda i \omega . X^{\prime} i \omega\right) J$
proof -
define $M$ where $M \equiv$ measure-pmf (Pi-pmf Id $p$ )
interpret prob-space $M$ using $M$-def prob-space-measure-pmf by blast
have indep-vars ( $\lambda$-. discrete) $(\lambda i x$. restrict-dfl $x(f-‘\{i\} \cap I) d)\left(f^{\prime} I\right)$
unfolding $M$-def by (rule indep-vars-restrict ${ }^{\prime}[$ OF assms(1)])
hence indep-vars $M^{\prime}\left(\lambda i \omega . X^{\prime} i(\right.$ restrict-dfl $\left.\omega(f-‘\{i\} \cap I) d)\right)(f$ ‘ $I)$
using assms(4)
by (intro indep-vars-compose2 [where $Y=X^{\prime}$ and $N=M^{\prime}$ and $M^{\prime}=\lambda$-. discrete])
(auto simp:assms(3))
hence indep-vars $M^{\prime}\left(\lambda i \omega . X^{\prime} i \omega\right)\left(f^{\prime} I\right)$
using assms(2)[symmetric]
by (simp add:assms(3) cong:indep-vars-cong)
thus ?thesis
unfolding $M$-def using assms(3) by simp
qed
lemma
fixes $f::{ }^{\prime} b \Rightarrow\left({ }^{\prime} c::\{\right.$ second-countable-topology,banach,real-normed-field $\left.\}\right)$
assumes finite $I$
assumes $i \in I$
assumes integrable (measure-pmf ( $M i$ )) $f$
shows integrable-Pi-pmf-slice: integrable (Pi-pmf I d M) $(\lambda x . f(x i))$
and expectation-Pi-pmf-slice: integral ${ }^{L}$ (Pi-pmf IdM) $(\lambda x . f(x i))=$ integral $^{L}$ (Mi) $f$
proof -
have a:distr (Pi-pmf I d M) (Mi) $(\lambda \omega . \omega i)=\operatorname{distr}($ Pi-pmf I d M) discrete ( $\lambda \omega . \omega i$ )
by (rule distr-cong, auto)
have b: measure-pmf.random-variable ( $M i$ ) borel $f$
using $\operatorname{assms}(3)$ by $\operatorname{simp}$
have $c$ :integrable (distr (Pi-pmf Id M) (Mi) $(\lambda \omega . \omega i)) f$
using $\operatorname{assms}(1,2,3)$
by (subst a, subst map-pmf-rep-eq[symmetric], subst Pi-pmf-component, auto)

```
    show integrable (Pi-pmf I d M) (\lambdax.f (x i))
    by (rule integrable-distr[where f=f and M'=M i]) (auto intro:c)
    have integral L
(M i) (\lambda\omega.\omegai)) f
    using b by (intro integral-distr[symmetric], auto)
    also have ... = integral }\mp@subsup{}{}{L}(\mathrm{ map-pmf ( }\lambda\omega.\omega\mathrm{ i ) (Pi-pmf I d M)) f
    by (subst a, subst map-pmf-rep-eq[symmetric], simp)
    also have ... = integral }\mp@subsup{}{}{L}(Mi)
    using assms(1,2) by (simp add: Pi-pmf-component)
    finally show integral L}(\mathrm{ Pi-pmf I d M) ( }\lambdax.f(xi))=\mp@subsup{integral }{L}{L}(Mi)f\mathrm{ by simp
qed
This is an improved version of expectation-prod-Pi-pmf. It works for general normed fields instead of non-negative real functions .
lemma expectation-prod-Pi-pmf:
fixes \(f:: ' a \Rightarrow ' b \Rightarrow\left({ }^{\prime} c::\{\right.\) second-countable-topology,banach,real-normed-field \(\left.\}\right)\) assumes finite \(I\)
assumes \(\bigwedge i . i \in I \Longrightarrow\) integrable (measure-pmf \((M i))(f i)\)
shows integral \({ }^{L}\left(\right.\) Pi-pmf IdM) \(\left(\lambda x .\left(\prod i \in I . f i(x i)\right)\right)=\left(\prod i \in I\right.\). integral \({ }^{L}\) (Mi) (fi))
proof -
have a: prob-space.indep-vars (measure-pmf (Pi-pmf I d M)) ( \(\lambda\)-. borel) ( \(\lambda i \omega . f\) \(i(\omega i)) I\) by (intro prob-space.indep-vars-compose2[where \(Y=f\) and \(M^{\prime}=\lambda\)-. discrete] prob-space-measure-pmf indep-vars-Pi-pmf assms(1)) auto
have integral \({ }^{L}\left(\right.\) Pi-pmf IdM) \(\left(\lambda x .\left(\prod i \in I . f i(x i)\right)\right)=\left(\prod i \in I\right.\). integral \(^{L}\) \((\) Pi-pmf I d M) \((\lambda x . f i(x i)))\)
by (intro prob-space.indep-vars-lebesgue-integral prob-space-measure-pmf assms(1,2) a integrable-Pi-pmf-slice) auto
also have \(\ldots=\left(\prod i \in I\right.\). integral \(\left.{ }^{L}(M i)(f i)\right)\)
by (intro prod.cong expectation-Pi-pmf-slice assms(1,2)) auto
finally show ?thesis by simp
qed
lemma variance-prod-pmf-slice:
fixes \(f\) :: ' \(a \Rightarrow\) real
assumes \(i \in I\) finite \(I\)
assumes integrable (measure-pmf (Mi)) ( \(\lambda \omega . f \omega^{\wedge}\) Z \()\)
shows prob-space.variance (Pi-pmf I d M) \((\lambda \omega . f(\omega i))=\) prob-space.variance
(Mi) \(f\)
proof -
have a:integrable (measure-pmf ( \(M i\) i)) \(f\)
using assms(3) measure-pmf.square-integrable-imp-integrable by auto
have b: integrable (measure-pmf (Pi-pmf Id M)) \(\left(\lambda x .(f(x i))^{2}\right)\)
by (rule integrable-Pi-pmf-slice[OF assms(2) assms(1)], metis assms(3))
have \(c\) : integrable (measure-pmf (Pi-pmf I d M)) \((\lambda x .(f(x i)))\)
by (rule integrable-Pi-pmf-slice[OF assms(2) assms(1)], metis a)
```

have measure-pmf.expectation (Pi-pmf I d M) $\left(\lambda x .(f(x i))^{2}\right)-($ measure-pmf.expectation $(\text { Pi-pmf Id M) }(\lambda x . f(x i)))^{2}=$
measure-pmf.expectation $\left(\begin{array}{ll}M & )\left(\lambda x .(f x)^{2}\right)-(\text { measure-pmf.expectation }(M)\end{array}\right.$ i) $f)^{2}$ using assms abchen ((subst expectation-Pi-pmf-slice $[O F \operatorname{assms}(2,1)])$ ?, simp $)+$
thus ?thesis
using assms abche (simp add: measure-pmf.variance-eq)
qed
lemma Pi-pmf-bind-return:
assumes finite $I$
shows Pi-pmf Id $(\lambda i . M i \gg(\lambda x$. return-pmf $(f i x)))=$ Pi-pmf $I d^{\prime} M \gg$ ( $\lambda x$. return-pmf $(\lambda i$. if $i \in I$ then $f i(x i)$ else $d)$ )
using assms by (simp add: Pi-pmf-bind $\left[\right.$ where $\left.d^{\prime}=d\right\rceil$ )
lemma pmf-of-set-prod-eq:
assumes $A \neq\{ \}$ finite $A$
assumes $B \neq\{ \}$ finite $B$
shows pmf-of-set $(A \times B)=$ pair-pmf (pmf-of-set $A)(p m f$-of-set $B)$
proof -
have indicat-real $(A \times B)(i, j)=$ indicat-real $A i *$ indicat-real $B j$ for $i j$ by (cases $i \in A$; cases $j \in B$ ) auto
hence $p m f(p m f$-of-set $(A \times B))(i, j)=p m f(p a i r-p m f(p m f$-of-set $A)(p m f$-of-set B)) (i,j)
for $i j$ using assms by (simp add:pmf-pair)
thus ?thesis by (intro pmf-eqI) auto
qed
lemma split-pmf-mod-div':
assumes $a>(0::$ nat $)$
assumes $b>0$
shows map-pmf $(\lambda x .(x \bmod a, x$ div $a))(p m f$-of-set $\{. .<a * b\})=p m f$-of-set $(\{. .<a\} \times\{. .<b\})$
using assms by (intro map-pmf-of-set-bij-betw bij-betw-prod finite-lessThan) (simp add: lessThan-empty-iff)
lemma split-pmf-mod-div:
assumes $a>(0::$ nat $)$
assumes $b>0$
shows map-pmf $(\lambda x .(x \bmod a, x$ div $a))(p m f$-of-set $\{. .<a * b\})=$ pair-pmf (pmf-of-set $\{. .<a\})(p m f$-of-set $\{. .<b\})$
using assms by (auto intro!: pmf-of-set-prod-eq simp add:split-pmf-mod-div')
end

## 6 Pseudorandom Objects

```
theory Pseudorandom-Objects
    imports Universal-Hash-Families-More-Product-PMF
begin
```

This section introduces a combinator library for pseudorandom objects [3]. These can be thought of as PRNGs but with rigorous mathematical properties, which can be used to in algorithms to reduce their randomness usage.
Such an object represents a non-empty multiset, with an efficient mechanism to sample from it. They have a natural interpretation as a probability space (each element is selected with a probability proportional to its occurrence count in the multiset).
The following section will introduce a construction of k-independent hash families as a pseudorandom object. The AFP entry Expander_Graphs then follows up with expander walks as pseudorandom objects.

```
record 'a pseudorandom-object =
    pro-last :: nat
    pro-select :: nat => 'a
definition pro-size where pro-size S = pro-last S +1
definition sample-pro where sample-pro S = map-pmf (pro-select S) (pmf-of-set
{0..pro-last S})
declare [[coercion sample-pro]]
abbreviation pro-set where pro-set S \equiv set-pmf (sample-pro S)
lemma sample-pro-alt: sample-pro S = map-pmf (pro-select S) (pmf-of-set {..<pro-size
S})
    unfolding pro-size-def sample-pro-def
    using Suc-eq-plus1 atLeast0AtMost lessThan-Suc-atMost by presburger
lemma pro-size-gt-0: pro-size S>0
    unfolding pro-size-def by auto
lemma set-sample-pro: pro-set S = pro-select S'{..<pro-size S}
    using pro-size-gt-0 unfolding sample-pro-alt set-map-pmf
    by (subst set-pmf-of-set) auto
lemma set-pmf-of-set-sample-size[simp]:
    set-pmf (pmf-of-set {..<pro-size S}) = {..<pro-size S}
    using pro-size-gt-0 by (intro set-pmf-of-set) auto
lemma pro-select-in-set: pro-select S (x mod pro-size S) \in pro-set S
    unfolding set-sample-pro by (intro imageI) (simp add:pro-size-gt-0)
lemma finite-pro-set: finite (pro-set S)
```

unfolding set-sample-pro by (intro finite-imageI) auto

```
lemma integrable-sample-pro[simp]:
    fixes f :: ' }a>>'c::{\mathrm{ banach, second-countable-topology}
    shows integrable (measure-pmf (sample-pro S))f
    by (intro integrable-measure-pmf-finite finite-pro-set)
```

definition list-pro :: 'a list $\Rightarrow$ 'a pseudorandom-object where
list-pro $l s=($ pro-last $=$ length $l s-1$, pro-select $=(!) l s)$
lemma list-pro:
assumes $x s \neq[]$
shows sample-pro (list-pro xs) $=$ pmf-of-multiset $($ mset xs) $($ is $? L=? R)$
proof -
have $? L=$ map-pmf ((!) xs) $(p m f$-of-set $\{. .<$ length $x s\})$
using assms unfolding list-pro-def sample-pro-alt pro-size-def by simp
also have $\ldots=$ pmf-of-multiset (image-mset ((!) xs) (mset-set $\{. .<$ length $x s\})$ )
using assms by (subst map-pmf-of-set) auto
also have...$=$ ? $R$
by (metis map-nth mset-map mset-set-upto-eq-mset-upto)
finally show ?thesis by simp
qed
lemma list-pro-2:
assumes $x s \neq[]$ distinct $x s$
shows sample-pro (list-pro $x s)=$ pmf-of-set $($ set $x s)($ is ? $L=? R)$
proof -
have $? L=\operatorname{map-pmf}((!) x s)(p m f$-of-set $\{. .<$ length $x s\})$
using assms unfolding list-pro-def sample-pro-alt pro-size-def by simp
also have $\ldots=p m f$-of-set ((!) xs ' $\{. .<$ length $x s\}$ )
using assms nth-eq-iff-index-eq by (intro map-pmf-of-set-inj inj-onI) auto
also have...$=$ ? $R$
by (intro arg-cong[where $f=p m f$-of-set $]$ ) (metis atLeast-upt list.set-map map-nth)
finally show?thesis by simp
qed
lemma list-pro-size:
assumes $x s \neq[]$
shows pro-size (list-pro xs) $=$ length $x s$
using assms unfolding pro-size-def list-pro-def by auto
lemma list-pro-set:
assumes $x s \neq[]$
shows pro-set (list-pro $x s$ ) $=$ set $x s$
proof -
have (!) xs ' $\{. .<$ length $x s\}=$ set $x s$ by (metis atLeast-upt list.set-map map-nth)
thus ?thesis unfolding set-sample-pro list-pro-size[OF assms] by (simp add:list-pro-def)
definition nat-pro :: nat $\Rightarrow$ nat pseudorandom-object where

$$
\text { nat-pro } n=(\text { pro-last }=n-1, \text { pro-select }=i d)
$$

lemma nat-pro-size:
assumes $n>0$
showspro-size (nat-pro $n$ ) $=n$
using assms unfolding nat-pro-def pro-size-def by auto

## lemma nat-pro:

assumes $n>0$
shows sample-pro (nat-pro $n$ ) $=$ pmf-of-set $\{. .<n\}$
unfolding sample-pro-alt nat-pro-size[OF assms] by (simp add:nat-pro-def)
lemma nat-pro-set:
assumes $n>0$
shows pro-set (nat-pro $n$ ) $=\{. .<n\}$
using assms unfolding nat-pro[OF assms] by (simp add: lessThan-empty-iff)

```
fun count-zeros :: nat \(\Rightarrow\) nat \(\Rightarrow\) nat where
    count-zeros \(0 k=0\)
    count-zeros (Suc n) \(k=(\) if odd \(k\) then 0 else \(1+\) count-zeros \(n\) ( \(k\) div 2) \()\)
```

lemma count-zeros-iff: $j \leq n \Longrightarrow$ count-zeros $n k \geq j \longleftrightarrow$ 2^j dvd $k$
proof (induction $j$ arbitrary: $n k$ )
case 0
then show ?case by simp
next
case (Suc j)
then obtain $n^{\prime}$ where $n$-def: $n=$ Suc $n^{\prime}$ using Suc-le-D by presburger
show ?case using Suc unfolding $n$-def by auto
qed
lemma count-zeros-max:
count-zeros $n k \leq n$
by (induction $n$ arbitrary: $k$ ) auto
definition geom-pro :: nat $\Rightarrow$ nat pseudorandom-object where
geom-pro $n=($ pro-last $=2$ 2 $n-1$, pro-select $=$ count-zeros $n$ )
lemma geom-pro-size: pro-size (geom-pro $n$ ) $=2$ 2 $n$
unfolding geom-pro-def pro-size-def by simp
lemma geom-pro-range: pro-set (geom-pro $n$ ) $\subseteq\{$..n $\}$
using count-zeros-max unfolding sample-pro-alt unfolding geom-pro-def by auto
lemma geom-pro-prob:
measure (sample-pro (geom-pro $n)$ ) $\{\omega . \omega \geq j\}=$ of-bool $(j \leq n) /$ 2^j (is ? $L=$ ?R)
proof (cases $j \leq n$ )
case True
have $a:\{. .<(2 \widehat{2})::$ nat $\} \neq\{ \}$
by (simp add: lessThan-empty-iff)
have $b:$ finite $\{. .<(2 \widehat{ }$ ) $::$ nat $\}$ by simp
define $f::$ nat $\Rightarrow$ nat where $f=(\lambda x . x * 2 \wedge j)$
have d:inj-on $f\left\{. .<\mathcal{Z}^{\wedge}(n-j)\right\}$ unfolding $f$-def by (intro inj-onI) simp
have $e: 2 \uparrow \mathfrak{j}>(0::$ nat $)$ by simp

```
have \(y \in f^{\prime}\left\{. .<\mathcal{Z}^{\wedge}(n-j)\right\} \longleftrightarrow y \in\left\{x . x<\mathcal{Z}^{\wedge} n \wedge \mathcal{Z}^{\wedge} j d v d x\right\}\) for \(y::\) nat
proof -
    have \(y \in f\) ' \(\left\{. .<\mathcal{Z}^{\wedge}(n-j)\right\} \longleftrightarrow\left(\exists x . x<2^{\wedge}(n-j) \wedge y=\mathcal{Z}^{\wedge} j * x\right)\)
        unfolding \(f\)-def by auto
```



```
        using \(e\) by simp
    also have \(\ldots \longleftrightarrow\left(\exists x .2\right.\) 2 \(\left.j * x<\mathscr{2}^{\wedge} n \wedge y=\mathcal{Z}^{\wedge} j * x\right)\)
        using True by (subst power-add[symmetric]) simp
    also have \(\ldots \longleftrightarrow\left(\exists x . y<2\right.\) 〔\(\left.n \wedge y=x * \mathcal{Z}^{\wedge} j\right)\)
        by (metis Groups.mult-ac(2))
```



```
    finally show?thesis by simp
qed
```

hence $c: f$ ' $\left\{. .<\mathcal{Z}^{\wedge}(n-j)\right\}=\{x . x<2$ 2^n $\wedge$ 2^j dvd $x\}$ by auto
have $? L=$ measure $(p m f$-of-set $\{. .<2 \widehat{2}\})\{\omega$. count-zeros $n \omega \geq j\}$
unfolding sample-pro-alt geom-pro-size by (simp add:geom-pro-def)
also have $\ldots=$ real (card $\left\{x::\right.$ nat. $\left.x<\mathcal{Z}^{\wedge} n \wedge \mathcal{Z}^{\wedge} j d v d x\right\}$ ) / $2 \uparrow n$
by (simp add: measure-pmf-of-set[OF a b] count-zeros-iff[OF True])
(simp add:lessThan-def Collect-conj-eq)
also have $\ldots=\operatorname{real}\left(\operatorname{card}\left(f \cdot\left\{. .<\mathcal{Z}^{\wedge}(n-j)\right\}\right)\right) / 2 \vee n$
by ( $\operatorname{simp}$ add:c)
also have $\ldots=\operatorname{real}\left(\operatorname{card}\left(\left\{. .<\left(\mathcal{L}^{\wedge}(n-j):: n a t\right)\right\}\right)\right) / \mathcal{Z}^{\text {亿 }} n$
by (simp add: card-image[OF d])
also have ... $=$ ? $R$
using True by (simp add:frac-eq-eq power-add[symmetric])
finally show ?thesis by simp
next
case False
have set-pmf (sample-pro (geom-pro $n$ )) $\subseteq\{$...n\}
using geom-pro-range by simp

```
    hence ?L = measure (sample-pro (geom-pro n)) {}
    using False by (intro measure-pmf-cong) auto
    also have ... = ? R
    using False by simp
    finally show ?thesis
    by simp
qed
lemma geom-pro-prob-single:
    measure (sample-pro (geom-pro n)) {j}\leq1/ 2^j (is ?L }\leq\mathrm{ ? R)
proof -
    have ?L = measure (sample-pro (geom-pro n)) ({j..}-{j+1..})
        by (intro measure-pmf-cong) auto
    also have ... = measure (sample-pro (geom-pro n)) {j..} - measure (sample-pro
(geom-pro n)) {j+1..}
    by (intro measure-Diff) auto
    also have ... = measure (sample-pro (geom-pro n)) {\omega.\omega\geqj}-measure (sample-pro
(geom-pro n)) {\omega.\omega\geq(j+1)}
    by (intro arg-cong2[where f=(-)] measure-pmf-cong) auto
    also have ... =of-bool (j\leqn)*1/2^j - of-bool (j+1\leqn)/2^(j+1)
        unfolding geom-pro-prob by simp
    also have ... \leq 1/2^j - 0
        by (intro diff-mono) auto
    also have ... =? R by simp
    finally show ?thesis by simp
qed
```

definition prod-pro ::
'a pseudorandom-object $\Rightarrow$ 'b pseudorandom-object $\Rightarrow(' a \times ' b)$ pseudorandom-object
where
prod-pro $P Q=$
( pro-last $=$ pro-size $P *$ pro-size $Q-1$,
pro-select $=(\lambda k$. (pro-select $P(k$ mod pro-size $P)$, pro-select $Q$ ( $k$ div pro-size
P))) D
lemma prod-pro-size:
pro-size $($ prod-pro $P Q)=$ pro-size $P *$ pro-size $Q$
unfolding prod-pro-def by (subst pro-size-def) (simp add:pro-size-gt-0)
lemma prod-pro:
sample-pro (prod-pro $P Q)=$ pair-pmf (sample-pro $P)($ sample-pro $Q)($ is $? L=$
?R)
proof -
let $? p=$ pro-size $P$
let $? q=$ pro-size $Q$
have ? $L=$ map-pmf $(\lambda k$. (pro-select $P(k$ mod ?p),pro-select $Q(k$ div ?p $)))$
(pmf-of-set $\{. .<? p * ? q\})$
unfolding sample-pro-alt prod-pro-size by (simp add:prod-pro-def)
also have $\ldots=\operatorname{map-pmf}($ map-prod $($ pro-select $P)($ pro-select $Q))$
$(\operatorname{map}-p m f(\lambda k .(k \bmod ? p, k$ div ? $p))(p m f$-of-set $\{. .<? p * ? q\}))$
unfolding map-pmf-comp by simp
also have $\ldots=$ ? $R$
unfolding split-pmf-mod-div[OF pro-size-gt-0 pro-size-gt-0] sample-pro-alt map-prod-def
map-pair
by $\operatorname{simp}$
finally show ?thesis by simp
qed
lemma prod-pro-set:
pro-set $($ prod-pro $P Q)=$ pro-set $P \times$ pro-set $Q$
unfolding prod-pro set-pair-pmf by simp
end

## 7 K-Independent Hash Families as Pseudorandom Objects

```
theory Pseudorandom-Objects-Hash-Families
    imports
        Pseudorandom-Objects
        Finite-Fields.Find-Irreducible-Poly
        Carter-Wegman-Hash-Family
        Universal-Hash-Families-More-Product-PMF
        begin
        hide-const (open) Numeral-Type.mod-ring
        hide-const (open) Divisibility.prime
        hide-const (open) Isolated.discrete
definition hash-space' ::
    ('a,'b) idx-ring-enum-scheme }=>\mathrm{ nat }=>('c,'d) pseudorandom-object-scheme
    #(nat }\mp@subsup{=>}{}{\prime}c) pseudorandom-objec
    where hash-space' R k S=(
        l
            pro-last = idx-size R `k-1,
            pro-select = ( \lambdax i.
                pro-select S
                (idx-enum-inv R (poly-eval R (poly-enum R k x) (idx-enum R i)) mod pro-size
S))
    D)
```

lemma prod-pmf-of-set:
assumes finite $A$ finite $B A \neq\{ \} B \neq\{ \}$
shows pmf-of-set $(A \times B)=$ pair-pmf $(p m f$-of-set $A)(p m f$-of-set $B)($ is $? L=$
?R)

```
proof (rule pmf-eqI)
    fix }
    have pmf ?L x = indicator }(A\timesB)x/\operatorname{real}(\operatorname{card}(A\timesB)
        using assms by (intro pmf-of-set) auto
    also have ... = (indicator A (fst x)/ real (card A)) * (indicator B (snd x)/ real
(card B))
    unfolding card-cartesian-product of-nat-mult by (simp add: indicator-times)
    also have ... = pmf (pmf-of-set A) (fst x) * pmf (pmf-of-set B) (snd x)
        by (intro arg-cong2[where f=(*)] pmf-of-set[symmetric] assms)
    also have ... = pmf ? R x
        unfolding pmf-pair[symmetric] by auto
    finally show pmf ?L x = pmf ?R x by simp
qed
lemma hash-prob-single':
    assumes field F finite (carrier F)
    assumes }x\in\mathrm{ carrier F
    assumes 1<n
    shows measure (pmf-of-set (bounded-degree-polynomials F n)) {\omega. ring.hash F x
\omega=y}=
    of-bool (y\in carrier F)/(real (card (carrier F))) (is ?L = ?R)
proof (cases y f carrier F)
    case True
    have ? L = \mathcal{P}(\omega\mathrm{ in pmf-of-set (bounded-degree-polynomials F n). ring.hash F x }\omega=\mp@code{}|=\mp@code{l}
= y) by simp
    also have ... = 1 / (real (card (carrier F))) by (intro hash-prob-single assms
conjI True)
    also have ... = ?R using True by simp
    finally show ?thesis by simp
next
    case False
    interpret field F using assms by simp
    have fin-carr: finite (carrier F) using assms by simp
    note S = non-empty-bounded-degree-polynomials fin-degree-bounded[OF fin-carr]
    let ?S = bounded-degree-polynomials F n
    have hash xf\not=y if f\in?S for f
    proof -
    have hash xf \in carrier F
            using that unfolding hash-def bounded-degree-polynomials-def
            by (intro eval-in-carrier assms) (simp add: polynomial-incl univ-poly-carrier)
            thus ?thesis using False by auto
    qed
    hence ?L = measure (pmf-of-set (bounded-degree-polynomials F n)) {}
        using}S\mathrm{ by (intro measure-eq-AE AE-pmfI) simp-all
    also have ... = ?R using False by simp
    finally show ?thesis by simp
qed
```

lemma hash-k-wise-indep':
assumes field $F \wedge$ finite (carrier $F$ )
assumes $1 \leq n$
shows prob-space.k-wise-indep-vars (pmf-of-set (bounded-degree-polynomials $F$ n)) $n$
( $\lambda$-. discrete) (ring.hash $F)($ carrier $F)$
by (intro prob-space.k-wise-indep-vars-compose[OF - hash-k-wise-indep[OF assms]] prob-space-measure-pmf) auto
lemma hash-space':
fixes $R::\left({ }^{\prime} a,{ }^{\prime} b\right)$ idx-ring-enum-scheme
assumes enum $_{C} R$ field $_{C} R$
assumes pro-size $S$ dvd order (ring-of $R$ )
assumes $I \subseteq\{. .<$ order (ring-of $R)\}$ card $I \leq k$
shows map-pmf $(\lambda f .(\lambda i \in I$. fi) $)$ (sample-pro (hash-space' $R k S)$ ) $=$ prod-pmf $I$
( $\lambda$-. sample-pro $S$ )
(is ? $L=? R$ )
proof $($ cases $I=\{ \})$
case False
let $? b=i d x$-size $R$
let $? s=$ pro-size $S$
let ?t $=$ ?b div?s
let ? $g=\lambda x i$. poly-eval $R($ poly-enum $R k x)(i d x$-enum $R i)$
let ?f $=\lambda$ x. pro-select $S$ (idx-enum-inv $R x$ mod ?s)
let ? $R$-pmf $=p m f$-of-set $($ carrier $($ ring-of $R))$
let $? S=\{x s \in$ carrier (poly-ring (ring-of $R$ )). length $x s \leq k\}$
let $? T=p m f$-of-set (bounded-degree-polynomials (ring-of $R$ ) $k$ )
interpret field ring-of $R$ using $\operatorname{assms}(2)$ unfolding field $C_{C}$-def by auto
have ring-c: ring $_{C} R$ using field-c-imp-ring assms(2) by auto
note enum-c $=$ enum- $-D[O F \operatorname{assms}(1)]$
have fin-carr: finite (carrier (ring-of $R$ )) using enum-c by simp
have $0<$ card I using False assms(4) card-gt-0-iff finite-nat-iff-bounded by blast also have $\ldots \leq k$ using $\operatorname{assms}(5)$ by $\operatorname{simp}$
finally have $k-g t-0: k>0$ by simp
have $b$-gt- $0: ~ ? b>0$ unfolding enum-c(2) using fin-carr order-gt- 0 -iff-finite by blast
hence $t$-gt-0: ?t $>0$ using enum-c(2) assms(3) dvd-div-gt0 by simp
have $b$ - $k$-gt- $0: ? b^{\wedge} k>0$ using $b$-gt- 0 by $\operatorname{simp}$
have fin-I: finite $I$ using assms(4) finite-subset by auto
have inj: inj-on (idx-enum $R$ ) I
using assms(4) unfolding enum-c(2)
by (intro inj-on-subset[OF bij-betw-imp-inj-on[OF enum-c(3)]])
have card $(i d x$-enum $R$ ' $I) \leq k$
using assms(5) unfolding card-image[OF inj] by auto
hence prob-space.indep-vars ?T ( $\lambda$-. discrete) hash (idx-enum $R$ ' $I$ )
using assms(4) k-gt-0 fin-I bij-betw-apply[OF enum-c(3)] enum-c(2)
by (intro prob-space.k-wise-indep-vars-subset[OF - hash-k-wise-indep $]$
prob-space-measure-pmf conjI fin-carr field-axioms) auto
hence prob-space.indep-vars ?T $((\lambda$-. discrete $) \circ i d x$-enum $R)(\lambda x \omega$. eval $\omega$ (idx-enum $R x$ ) ) I
using inj unfolding hash-def
by (intro prob-space.indep-vars-reindex prob-space-measure-pmf) auto
hence indep: prob-space.indep-vars ?T ( $\lambda$-. discrete) ( $\lambda x \omega$. eval $\omega$ (idx-enum $R$ x)) $I$
by (simp add:comp-def)
have 0: pmf (map-pmf $(\lambda x$. $\lambda i \in I$. eval $x(i d x$-enum $R i)$ ? $T) \omega=p m f($ prod-pmf $I(\lambda-. ? R-p m f)) \omega$
(is ? L1 = ? R1) for $\omega$
proof (cases $\omega \in$ extensional $I$ )
case True
have ? L1 $=$ measure ? $T\{x .(\lambda i \in I$. eval $x($ idx-enum $R i))=\omega\}$
by (simp add:pmf-map vimage-def)
also have $\ldots=$ measure ? $T\{x .(\forall i \in I$. eval $x(i d x$-enum $R i)=\omega i)\}$
using True unfolding restrict-def extensional-def
by (intro arg-cong2[where $f=$ measure] refl Collect-cong) auto
also have $\ldots=\left(\prod i \in I\right.$. measure ? $T$ \{x. eval $x(i d x$-enum $\left.\left.R i)=\omega i\right\}\right)$
by (intro prob-space.split-indep-events[where $I=I$ and $p=$ ?T] prob-space-measure-pmf fin-I refl prob-space.indep-vars-compose2 $[O F$ - indep]) auto
also have $\ldots=\left(\prod i \in I\right.$. measure ?T \{x. hash (idx-enum $R$ i) $\left.\left.x=\omega i\right\}\right)$
unfolding hash-def by simp
also have $\ldots=\left(\prod i \in I\right.$. of-bool $(\omega i \in$ carrier (ring-of $\left.R)\right) /$ real (card (carrier ( ring-of $R$ )) )
using $k$-gt-0 assms(4) by (intro prod.cong refl hash-prob-single ${ }^{\prime}$ bij-betw-apply[OF enum-c (3)] fin-carr field-axioms) (auto simp:enum-c)
also have $\ldots=\left(\prod i \in I . p m f(p m f\right.$-of-set $($ carrier $($ ring-of $\left.R)))(\omega i)\right)$
using fin-carr carrier-not-empty by (simp add:indicator-def)
also have $\ldots=$ ? $R 1$
using True unfolding pmf-prod-pmf[OF fin-I] by simp
finally show? ?thesis by simp
next
case False
have ?L1 $=0$ using False unfolding pmf-eq-0-set-pmf set-map-pmf by auto
moreover have ? $R 1=0$
using False unfolding pmf-eq-O-set-pmf set-prod-pmf[OF fin-I] PiE-def by simp
ultimately show? thesis by simp
qed
have map-pmf $(\lambda x . \lambda i \in I$. ?g $x i)(p m f$-of-set $\{. .<? b \wedge k\})=$
map-pmf $(\lambda x . \lambda i \in I$. poly-eval $R x(i d x$-enum $R i)$ (map-pmf (poly-enum $R k)$

```
(pmf-of-set {..<?b^k}))
```

by (simp add:map-pmf-comp)
also have $\ldots=\operatorname{map-pmf}(\lambda x$. $\lambda i \in I$. poly-eval $R x(i d x$-enum $R i))(p m f$-of-set ?S)
using $b$-k-gt-0 by (intro arg-cong2[where $f=$ map-pmf] refl map-pmf-of-set-bij-betw bij-betw-poly-enum $\operatorname{assms}(1,2)$ field-c-imp-ring) blast+
also have $\ldots=\operatorname{map}-p m f(\lambda x . \lambda i \in I$. poly-eval $R x(i d x$-enum $R i)) ? T$
using $k$-gt-0 unfolding bounded-degree-polynomials-def
by (intro map-pmf-cong refl arg-cong[where $f=p m f$-of-set $]$ restrict-ext ring-c) auto
also have $\ldots=\operatorname{map-pmf}(\lambda x$. $\lambda i \in I$. eval $x(i d x$-enum $R i)) ? T$
using non-empty-bounded-degree-polynomials fin-degree-bounded[OF fin-carr] assms(4)
by (intro map-pmf-cong poly-eval refl restrict-ext ring-c bij-betw-apply[OF enum-c(3)])
(auto simp add:bounded-degree-polynomials-def ring-of-poly[OF ring-c] enum-c(2))
also have $\ldots=$ prod-pmf $I(\lambda-$. ? $R$-pmf) $($ is ? $L 1=? R 1)$
by (intro pmf-eqI 0)
finally have $0: \operatorname{map-pmf}(\lambda x . \lambda i \in I$. ?g $x i)(p m f$-of-set $\{. .<? b \wedge k)=p r o d-p m f$ $I(\lambda-$. ?R-pmf)
by $\operatorname{simp}$
have 1: map-pmf $(\lambda x . x$ mod ? s $)(p m f$-of-set $\{. .<? b\})=p m f$-of-set $\{. .<? s\}$ (is ? L $1=$ ? R1)
proof -
have ?L1 $=$ map-pmf fst $(\operatorname{map}-p m f(\lambda x .(x \bmod ? s, x$ div ?s) $)(p m f-o f-s e t$ $\{. .<? s * ? t\}))$
using $\operatorname{assms}(3)$ by (simp add:map-pmf-comp enum-c(2))
also have $\ldots=\operatorname{map}-p m f f s t(p m f-o f-s e t(\{. .<? s\} \times\{. .<? t\}))$
using pro-size-gt-0 t-gt-0 lessThan-empty-iff finite-lessThan
by (intro arg-cong2[where $f=$ map-pmf] refl map-pmf-of-set-bij-betw bij-betw-prod) force+
also have $\ldots=$ map-pmffst $($ pair-pmf $(p m f$-of-set $\{. .<? s\})(p m f$-of-set $\{. .<? t\}))$
using pro-size-gt-0 t-gt-0 by (intro arg-cong2[where $f=$ map-pmf] prod-pmf-of-set refl) auto
also have $\ldots=p m f$-of-set $\{. .<? s\}$ using map-fst-pair-pmf by blast
finally show? ?thesis by simp
qed
have map-pmf ?f ? $R-p m f=\operatorname{map-pmf}(\lambda x$. pro-select $S(x \bmod ? s))(\operatorname{map-pmf}$ (idx-enum-inv $R$ ) ?R-pmf)
by (simp add:map-pmf-comp)
also have $\ldots=\operatorname{map-pmf}(\lambda x$. pro-select $S(x \bmod ? s))(p m f$-of-set $\{. .<? b\})$
using enum-cD(1,2,4)[OF assms(1)] carrier-not-empty
by (intro arg-cong2[where $f=$ map-pmf] refl map-pmf-of-set-bij-betw) auto
also have $\ldots=\operatorname{map-pmf}($ pro-select $S)($ map-pmf $(\lambda x . x$ mod ?s) (pmf-of-set $\{. .<? b\})$ )
by (simp add:map-pmf-comp)
also have $\ldots=$ sample-pro $S$ unfolding sample-pro-alt 1 by simp
finally have 2:map-pmf ?f ?R-pmf $=$ sample-pro $S$ by simp
have ? $L=\operatorname{map-pmf}(\lambda x . \lambda i \in I$. ?f $(? g \quad x i))\left(p m f\right.$-of-set $\left.\left\{. .<? b \not{ }^{\prime} k\right\}\right)$
using $b$ - $k$-gt- 0 unfolding sample-pro-alt hash-space'-def pro-size-def
by (simp add: map-pmf-comp del:poly-eval.simps)
also have $\ldots=\operatorname{map-pmf}(\lambda f . \lambda i \in I$. ?f $(f i))($ map-pmf $(\lambda x . \lambda i \in I$. ?g $x i)$
(pmf-of-set $\left.\left\{. .<? b^{\wedge} k\right\}\right)$ )
unfolding map-pmf-comp by (intro arg-cong2[where $f=m a p-p m f]$ refl re-strict-ext ext) simp
also have $\ldots=$ prod-pmf $I(\lambda$-. map-pmf ?f $($ pmf-of-set $(\operatorname{carrier}($ ring-of $R))))$
unfolding 0
by (simp add:map-pmf-def Pi-pmf-bind-return[OF fin-I, where $\left.d^{\prime}=u n d e f i n e d\right]$ restrict-def)
also have $\ldots=$ ? $R$ unfolding 2 by simp
finally show? thesis by simp
next
case True
have ? $L=$ map-pmf ( $\lambda f$ i. undefined) $($ sample-pro (hash-space $R k S)$ )
using True by (intro map-pmf-cong refl) auto
also have $\ldots=$ return-pmf ( $\lambda f$. undefined) unfolding map-pmf-const by simp
also have $\ldots=? R$ using True by simp
finally show ? $L=? R$ by $\operatorname{simp}$
qed
lemma hash-space'-range:
pro-select (hash-space ${ }^{\prime} R k S$ ) $i j \in$ pro-set $S$
unfolding hash-space'-def by (simp add: pro-select-in-set)
definition hash-pro ::
nat $\Rightarrow$ nat $\Rightarrow\left({ }^{\prime} a,{ }^{\prime} b\right)$ pseudorandom-object-scheme $\Rightarrow\left(n a t \Rightarrow{ }^{\prime} a\right)$ pseudoran-dom-object
where hash-pro $k d S=($
let $(p, j)=$ split-power $($ pro-size $S)$;
$l=\max j($ floorlog $p(d-1))$
in hash-space $\left.{ }^{\prime}\left(G F\left(p^{\wedge} l\right)\right) k S\right)$
definition hash-pro-spmf ::
nat $\Rightarrow$ nat $\Rightarrow\left({ }^{\prime} a,{ }^{\prime} b\right)$ pseudorandom-object-scheme $\Rightarrow\left(n a t \Rightarrow{ }^{\prime} a\right)$ pseudoran-
dom-object spmf
where hash-pro-spmf $k d S=$
do \{
let $(p, j)=$ split-power $($ pro-size $S)$;
let $l=\max j($ floorlog $p(d-1))$;
$R \leftarrow G F_{R}\left(p^{\wedge} l\right) ;$
return-spmf (hash-space ${ }^{\prime} R k S$ )
\}
definition hash-pro-pmf ::
nat $\Rightarrow$ nat $\Rightarrow\left({ }^{\prime} a,{ }^{\prime} b\right)$ pseudorandom-object-scheme $\Rightarrow\left(n a t \Rightarrow{ }^{\prime} a\right)$ pseudoran-
dom-object pmf
where hash-pro-pmf $k d S=$ map-pmf the (hash-pro-spmf $k d S)$
syntax
-FLIPBIND $\quad::\left({ }^{\prime} a \Rightarrow{ }^{\prime} b\right) \Rightarrow^{\prime} c \Rightarrow{ }^{\prime} b \quad(\mathbf{i n f i x r}=\ll 54)$

## translations

- FLIPBIND $f g=>g \gg f$
context
fixes $S$
fixes $d::$ nat
fixes $k::$ nat
assumes size-prime-power: is-prime-power (pro-size $S$ )
begin
private definition $p$ where $p=$ fst (split-power (pro-size $S$ ))
private definition $j$ where $j=$ snd (split-power (pro-size $S$ ))
private definition $l$ where $l=\max j($ floorlog $p(d-1)$ )
private lemma split-power: $(p, j)=$ split-power $($ pro-size $S)$
using $p$-def $j$-def by auto
private lemma hash-sample-space-alt: hash-pro $k d S=h a s h-s p a c e ' ~(G F(p \wedge)) k$ $S$
unfolding hash-pro-def split-power[symmetric] by (simp add:j-def l-def Let-def)
private lemma $p$-prime : prime $p$ and $j$-gt- $0: j>0$
proof -
obtain $q r$ where 0:pro-size $S=q \widehat{r}$ and $q$-prime: prime $q$ and $r$-gt- $0: r>0$
using size-prime-power is-prime-power-def by blast
have $(p, j)=$ split-power ( $q \widehat{r}$ ) unfolding split-power 0 by simp
also have $\ldots=(q, r)$ by (intro split-power-prime $q$-prime r-gt-0)
finally have $(p, j)=(q, r)$ by simp
thus prime $p j>0$ using $q$-prime $r$-gt- 0 by auto
qed
private lemma l-gt-0: $l>0$
unfolding $l$-def using $j$-gt-0 by simp
private lemma prime-power: is-prime-power ( $p$ ^l)
using $p$-prime l-gt-0 unfolding is-prime-power-def by auto
lemma hash-in-hash-pro-spmf: hash-pro $k d S \in$ set-spmf (hash-pro-spmf k d $S$ ) using GF-in-GF-R[OF prime-power]
unfolding hash-pro-def hash-pro-spmf-def split-power[symmetric] l-def by (auto simp add:set-bind-spmf)

```
lemma lossless-hash-pro-spmf: lossless-spmf (hash-pro-spmf k d S)
proof -
    have lossless-spmf (G\mp@subsup{F}{R}{}(\mp@subsup{p}{}{`}l)) by (intro galois-field-random-1 prime-power)
    thus ?thesis unfolding hash-pro-spmf-def split-power[symmetric] l-def by simp
qed
lemma hashp-eq-hash-pro-spmf: set-pmf (hash-pro-pmf k d S) = set-spmf (hash-pro-spmf
kdS)
    unfolding hash-pro-pmf-def using lossless-imp-spmf-of-pmf[OF lossless-hash-pro-spmf]
    by (metis set-spmf-spmf-of-pmf)
lemma hashp-in-hash-pro-spmf:
    assumes x fet-pmf (hash-pro-pmf kd S)
    shows x \in set-spmf (hash-pro-spmf k d S)
    using hashp-eq-hash-pro-spmf assms by auto
lemma hash-pro-in-hash-pro-pmf: hash-pro k d S \in set-pmf (hash-pro-pmf k d S)
    unfolding hashp-eq-hash-pro-spmf by (intro hash-in-hash-pro-spmf)
lemma hash-pro-spmf-distr:
    assumes s f set-spmf (hash-pro-spmf k d S)
    assumes I\subseteq{..<d} card I\leqk
    shows map-pmf (\lambdaf. (\lambdai\inI.fi)) (sample-pro s) = prod-pmf I ( }\lambda\mathrm{ -. sample-pro
S)
proof -
    have (d-1)< p^floorlog p (d-1)
        using floorlog-leD prime-gt-1-nat[OF p-prime] by simp
    hence d}\leq\mp@subsup{p}{}{`}\mathrm{ ^loorlog p(d-1) by (cases d) auto
    also have ... \leq p`l
        using prime-gt-O-nat[OF p-prime] unfolding l-def by (intro power-increasing)
auto
    finally have 0:d\leqp`l by simp
    obtain R where R-in: R set-spmf (GF R (p^l)) and s-def:s=hash-space' R
kS
    using assms(1) unfolding hash-pro-spmf-def split-power[symmetric] l-def
    by (auto simp add:set-bind-spmf)
    have 1: order (ring-of R)= p^l
        using galois-field-random-1(1)[OF prime-power R-in] by auto
    have I\subseteq{..<d} using assms by auto
    also have ...\subseteq{..<order (ring-of R)} using 0 unfolding 1 by auto
    finally have I\subseteq{..<order (ring-of R)} by simp
    moreover have j\leql unfolding l-def by auto
    hence pro-size S dvd order (ring-of R)
        unfolding 1 split-power-result[OF split-power] by (intro le-imp-power-dvd)
    ultimately show ?thesis
        using galois-field-random-1 (1)[OF prime-power R-in] assms(3)
        unfolding s-def by (intro hash-space') simp-all
qed
```

```
lemma hash-pro-spmf-component:
    assumes s f set-spmf (hash-pro-spmf k d S)
    assumes i<dk>0
    shows map-pmf (\lambdaf.fi)(sample-pro s)= sample-pro S (is ?L =?R)
proof -
    have ?L = map-pmf ( }\lambdaf.fi)(\mathrm{ map-pmf ( }\lambdaf.(\lambdai\in{i}.fi))(sample-pro s)
        using assms(1) unfolding map-pmf-comp by (intro map-pmf-cong refl) auto
    also have ... = map-pmf (\lambdaf.fi)(prod-pmf {i} (\lambda-. sample-pro S))
        using assms by (subst hash-pro-spmf-distr[OF assms(1)]) auto
    also have ... = ?R by (subst Pi-pmf-component) auto
    finally show ?thesis by simp
qed
lemma hash-pro-spmf-indep:
    assumes s \in set-spmf (hash-pro-spmf k d S)
    assumes I\subseteq{..<d} card I\leqk
    shows prob-space.indep-vars (sample-pro s) (\lambda-. discrete) (\lambdai\omega.\omega i)I
proof (rule measure-pmf.indep-vars-pmf[OF refl])
    fix }x
    assume a:J\subseteqI
    have 0:J\subseteq{..<d} using a assms(2) by auto
    have card J \leq card I using finite-subset[OF assms(2)] by (intro card-mono a)
auto
    also have ...\leqk using assms(3) by simp
    finally have 1: card J\leqk by simp
    let ?s = sample-pro s
    have 2: 0<k if }x\inJ\mathrm{ for }
    proof -
        have 0<card J using 0 that card-gt-0-iff finite-nat-iff-bounded by auto
        also have ... \leqk using 1 by simp
        finally show ?thesis by simp
    qed
    have measure ?s {\omega.\forallj\inJ.\omega j=x j}= measure (map-pmf ( }\lambda\omega.\lambdaj\inJ.\omegaj)?s
{\omega.}\forallj\inJ.\omegaj=xj
    by auto
    also have ... = measure (prod-pmf J (\lambda-. sample-pro S)) (Pi J (\lambdaj. {x j}))
        unfolding hash-pro-spmf-distr[OF assms(1) 0 1] by (intro arg-cong2[where
f=measure]) (auto simp:Pi-def)
    also have ... = (\prodj\inJ. measure (sample-pro S) {x j})
    using finite-subset[OF a] finite-subset[OF assms(2)] by (intro measure-Pi-pmf-Pi)
auto
    also have ... = (\prodj\inJ. measure (map-pmf (\lambda\omega.\omega j) ?s) {x j})
        using 012 by (intro prod.cong arg-cong2[where f=measure] refl
            arg-cong[where f=measure-pmf] hash-pro-spmf-component[OF assms(1),
symmetric]) auto
    also have ... = (\prodj\inJ. measure ?s {\omega.\omega j=x j}) by (simp add:vimage-def)
```

finally show measure ?s $\{\omega . \forall j \in J . \omega j=x j\}=\left(\prod j \in J\right.$. measure-pmf.prob ?s $\{\omega \cdot \omega j=x j\}$ )
by simp
qed
lemma hash-pro-spmf-k-indep:
assumes $s \in$ set-spmf (hash-pro-spmf $k d S$ )
shows prob-space.k-wise-indep-vars (sample-pro s) $k$ ( $\lambda$-. discrete) ( $\lambda i \omega . \omega i$ ) $\{. .<d\}$
using hash-pro-spmf-indep[OF assms]
unfolding prob-space.k-wise-indep-vars-def[OF prob-space-measure-pmf] by auto
private lemma hash-pro-spmf-size-aux:
assumes $s \in$ set-spmf (hash-pro-spmf $k d S$ )
shows pro-size $s=\left(p^{\wedge} l\right) \wedge k$ (is ? $\left.L=? R\right)$
proof -
obtain $R$ where $R$-in: $R \in \operatorname{set}$-spmf $\left(G F_{R}\left(p^{\wedge} l\right)\right)$ and $s$-def: $s=$ hash-space ${ }^{\prime} R$ $k S$
using assms(1) unfolding hash-pro-spmf-def split-power[symmetric] l-def
by (auto simp add:set-bind-spmf)
have 1: order (ring-of $R)=p^{\wedge} l$ and ec: enum $_{C} R$
using galois-field-random-1 (1)[OF prime-power $R$-in] by auto
have ? $L=$ idx-size $R^{\wedge} k-1+1$
unfolding $s$-def pro-size-def hash-space'-def by simp
also have $\ldots=\left(\left(p^{\wedge} l\right) \uparrow k-1\right)+1$
using 1 enum- $c D(2)[O F e c]$ by simp
also have $\ldots=\left(p^{\wedge} l\right)^{\wedge} k$ using prime-gt- 0 -nat $[O F$ p-prime $]$ by simp
finally show?thesis by simp
qed
lemma floorlog-alt-def:
floorlog $b a=($ if $1<b$ then nat $\lceil\log ($ real $b)($ real $a+1)\rceil$ else 0$)$
proof (cases $a>0 \wedge 1<b)$
case True
have 1: log (real b) (real $a+1)>0$ using True by (subst zero-less-log-cancel-iff) auto
have $a<\operatorname{real} a+1$ by $\operatorname{simp}$
also have $\ldots=b$ powr $(\log b($ real $a+1))$ using True by simp
also have $\ldots \leq b$ powr $(\lceil\log b($ real $a+1)\rceil)$
using True by (intro iffD2[OF powr-le-cancel-iff]) auto
also have $\ldots=b \operatorname{powr}($ real $($ nat $\lceil\log b($ real $a+1)\rceil))$
using 1 by (intro arg-cong2[where $f=($ powr $)]$ refl) linarith
also have $\ldots=b{ }^{\wedge}$ nat $\lceil\log$ (real b) (real $\left.a+1)\right\rceil$ using True by (subst powr-realpow) auto
finally have $a<b$ へ nat $\lceil\log ($ real $b)($ real $a+1)\rceil$ by simp
hence 0:floorlog basnat「log (real b) (real a+1)ๆ using True by (intro
have $b^{\wedge}($ nat $\lceil\log b($ real $a+1)\rceil-1)=b$ powr (real $(n a t\lceil\log b($ real $a+1)\rceil$ - 1))
using True by (subst powr-realpow) auto
also have $\ldots=b$ powr $(\lceil\log b($ real $a+1)\rceil-1)$
using 1 by (intro arg-cong2 [where $f=($ powr $)]$ refl) linarith
also have $\ldots<b$ powr $(\log b($ real $a+1))$ using True by (intro powr-less-mono)
linarith+
also have $\ldots=$ real $(a+1)$ using True by simp
finally have $b^{\wedge}($ nat $\lceil\log ($ real $b)($ real $a+1)\rceil-1)<a+1$ by linarith
hence $b^{\wedge}($ nat $\lceil\log ($ real $b)($ real $a+1)\rceil-1) \leq a$ by simp
hence floorlog $b a \geq$ nat $\lceil\log$ (real b) (real $a+1$ ) $\rceil$ using True by (intro floor-
log-geI) auto
hence floorlog $b a=n a t\lceil l o g($ real $b)($ real $a+1)\rceil$ using 0 by linarith
also have $\ldots=($ if $1<b$ then nat $\lceil\log ($ real b) $($ real $a+1)\rceil$ else 0) using True
by $\operatorname{simp}$
finally show ?thesis by simp
next
case False
hence $a$-eq- $0: a=0 \vee \neg(1<b)$ by simp
thus ?thesis unfolding floorlog-def by auto
qed
lemma hash-pro-spmf-size:
assumes $s \in$ set-spmf (hash-pro-spmf kd $\operatorname{S}$ )
assumes $\left(p^{\prime}, j^{\prime}\right)=$ split-power (pro-size $S$ )
shows pro-size $s=\left(p^{\prime \wedge}\left(\max j^{\prime}\left(\text { floorlog } p^{\prime}(d-1)\right)\right)\right)^{\wedge} k$
unfolding hash-pro-spmf-size-aux[OF assms(1)] l-def p-def j-def using assms(2)
by (metis fst-conv snd-conv)
lemma hash-pro-spmf-size':
assumes $s \in$ set-spmf (hash-pro-spmf $k d S$ ) $d>0$
assumes $\left(p^{\prime}, j^{\prime}\right)=$ split-power (pro-size $S$ )
shows pro-size $s=\left(p^{\prime \wedge}\left(k * \max j^{\prime}\left(\right.\right.\right.$ nat $\left.\left.\left.\left\lceil\log p^{\prime} d\right\rceil\right)\right)\right)$
proof -
have pro-size $s=\left(p^{\wedge}(\max j(\text { floorlog } p(d-1)))\right)^{\wedge} k$
unfolding hash-pro-spmf-size-aux[OF assms(1)] l-def by simp
also have $\ldots=\left(p^{\wedge}(\max j(\text { nat }\lceil\log p(\operatorname{real}(d-1)+1)\rceil))\right)^{\wedge} k$
using prime-gt-1-nat[OF p-prime] by (simp add:floorlog-alt-def)
also have $\ldots=\left(p^{\wedge}(\max j(n a t\lceil\log p d\rceil))\right)^{\wedge} k$ using $\operatorname{assms}(2)$ by (subst of-nat-diff)
auto
also have $\ldots=p^{\wedge}(k * \max j($ nat $\lceil\log p d\rceil))$ by (simp add:ac-simps power-mult $[$ symmetric $\left.]\right)$
also have $\ldots=p^{\prime}\left(k * \max j^{\prime}\left(\right.\right.$ nat $\left.\left.\left\lceil\log p^{\prime} d\right\rceil\right)\right)$
using $\operatorname{assms}(3) p$-def $j$-def by (metis fst-conv snd-conv)
finally show ?thesis by simp
qed
lemma hash-pro-spmf-size-prime-power:

```
    assumes s \in set-spmf (hash-pro-spmf k d S)
    assumes k>0
    shows is-prime-power (pro-size s)
    unfolding hash-pro-spmf-size-aux[OF assms(1)] power-mult[symmetric] is-prime-power-def
    using p-prime mult-pos-pos[OF l-gt-0 assms(2)] by blast
    lemma hash-pro-smpf-range:
        assumes s\in set-spmf (hash-pro-spmf k d S)
        shows pro-select s i q}\in\mathrm{ pro-set S
proof -
    obtain R where R-in: R \in set-spmf (GF R ( p^l)) and s-def:s=hash-space' R
kS
    using assms(1) unfolding hash-pro-spmf-def split-power[symmetric] l-def
    by (auto simp add:set-bind-spmf)
    thus ?thesis
    unfolding s-def using hash-space'-range by auto
qed
lemmas hash-pro-size' = hash-pro-spmf-size'[OF hash-in-hash-pro-spmf]
lemmas hash-pro-size = hash-pro-spmf-size[OF hash-in-hash-pro-spmf]
lemmas hash-pro-size-prime-power = hash-pro-spmf-size-prime-power[OF hash-in-hash-pro-spmf]
lemmas hash-pro-distr = hash-pro-spmf-distr[OF hash-in-hash-pro-spmf]
lemmas hash-pro-component = hash-pro-spmf-component[OF hash-in-hash-pro-spmf]
lemmas hash-pro-indep = hash-pro-spmf-indep[OF hash-in-hash-pro-spmf]
lemmas hash-pro-k-indep = hash-pro-spmf-k-indep[OF hash-in-hash-pro-spmf]
lemmas hash-pro-range = hash-pro-smpf-range[OF hash-in-hash-pro-spmf]
lemmas hash-pro-pmf-size' = hash-pro-spmf-size'[OF hashp-in-hash-pro-spmf]
lemmas hash-pro-pmf-size = hash-pro-spmf-size[OF hashp-in-hash-pro-spmf]
lemmas hash-pro-pmf-size-prime-power = hash-pro-spmf-size-prime-power[OF hashp-in-hash-pro-spmf]
lemmas hash-pro-pmf-distr = hash-pro-spmf-distr[OF hashp-in-hash-pro-spmf]
lemmas hash-pro-pmf-component = hash-pro-spmf-component[OF hashp-in-hash-pro-spmf]
lemmas hash-pro-pmf-indep = hash-pro-spmf-indep[OF hashp-in-hash-pro-spmf]
lemmas hash-pro-pmf-k-indep = hash-pro-spmf-k-indep[OF hashp-in-hash-pro-spmf]
lemmas hash-pro-pmf-range = hash-pro-smpf-range[OF hashp-in-hash-pro-spmf]
end
```

bundle pseudorandom-object-notation
begin
notation hash-pro ( $\mathcal{H}$ )
notation hash-pro-spmf $\left(\mathcal{H}_{S}\right)$
notation hash-pro-pmf $\left(\mathcal{H}_{P}\right)$
notation list-pro ( $\mathcal{L}$ )
notation nat-pro $(\mathcal{N})$
notation geom-pro ( $\mathcal{G}$ )
notation prod-pro (infixr $\times_{P} 65$ )
end
bundle no-pseudorandom-object-notation
begin
no-notation hash-pro ( $\mathcal{H}$ )
no-notation hash-pro-spmf $\left(\mathcal{H}_{S}\right)$
no-notation hash-pro-pmf ( $\mathcal{H}_{P}$ )
no-notation list-pro $(\mathcal{L})$
no-notation nat-pro ( $\mathcal{N}$ )
no-notation geom-pro $(\mathcal{G})$
no-notation prod-pro (infixr $\times_{P} 65$ )
end
unbundle pseudorandom-object-notation
end

## References

[1] E. Karayel. Interpolation polynomials (in hol-algebra). Archive of Formal Proofs, Jan. 2022. https://isa-afp.org/entries/Interpolation_ Polynomials_HOL_Algebra.html, Formal proof development.
[2] M. Thorup and Y. Zhang. Tabulation based 5-universal hashing and linear probing. In Proceedings of the Meeting on Algorithm Engineering छ Expermiments, ALENEX '10, pages 62-76, USA, 2010. Society for Industrial and Applied Mathematics.
[3] S. P. Vadhan. Pseudorandomness. Foundations and Trends®in Theoretical Computer Science, 7(1-3):1-336, 2012.
[4] M. N. Wegman and J. L. Carter. New hash functions and their use in authentication and set equality. Journal of Computer and System Sciences, 22(3):265-279, 1981.

